2

AD-A214 043

BM/C 3 TECHNOLOGY (MULTI-GRAIN)

FINAL REPORT

29 SEPTEMBER 1989

STRATEGIC DEFENSE INITIATIVE ORGANIZATION
Office Of The Secretary Of Defense
Washington, DC 20301-7100

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY 2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION The Analytic Sciences Corporation (Prime Contractor) 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Strategic Defense Initiative Organization
6c. ADDRESS (City, State, and ZIP Code) 1700 N. Moore Street Suite 1800 Arlington, VA 22209	7b. ADDRESS (City, State, and ZIP Code) Room 1E149 The Pentagon Washington, D.C. 20301-7100
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Strategic Defense Initiative Organization 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Room 1E149 The Pentagon Washington, D.C. 20301-7100	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
BM/C3 Technology (Multi-Grain) Final Report (U)

12. PERSONAL AUTHOR(S)

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM 13Feb89 TO 15Sep89	1989, September, 29	447

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	SDS BATTLE MANAGEMENT, COMMAND, CONTROL AND COMMUNICATIONS, PARALLEL PROCESSORS TECHNOLOGY, BM/C3 ALGORITHMS, PROCESSOR SIZING, AUTOMATED SIZING TOOLS

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report presents a method to assess, reliably and rapidly, the performance of parallel (multi-grain) processing architectures for SDS BM/C3 requirements. The existing tools and approaches for making such estimates are either "back-of-the-envelope" approach, or a high fidelity event driven simulations. The processor sizing software tool PERM (Processor Ensemble Runtime Model) developed in this contract is intended to fill the gap between the extremes of very low fidelity and very high fidelity models. PERM explicitly models concurrent use of shared resources (memory, bus bandwidth, etc.) in a deterministic fashion and it provides a natural, menu-driven user interface for the construction of date structures and display of results. The methods in algorithm-to-hardware architecture mapping for parallel processing were demonstrated by mapping the AOA tracking algorithm on to the AOSP hardware as a testcase.

(Continued on reverse side)

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☒ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL CPT Stephen L. Johnson	22b. TELEPHONE (Include Area Code) (202) 693-1600 22c. OFFICE SYMBOL SDIO/ENA

DD Form 1473, JUN 86 Previous editions are obsolete.

89 11 02 074

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19. (Continued)

Section 1 provides the System Description for PERM. Section 2 is the PERM User's Guide. Section 3 presents the testcase. The performance data generated by PERM was analyzed. A discussion is also provided of the experience using PERM both strengths and weaknesses.

FOREWORD

This technical report was prepared by SPARTA, Inc., Teledyne Brown Engineering, and The Analytic Sciences Corporation (TASC), for SDIO, as specified by Task Order 14, Data Item Instructions, Sequence Number A250, Contract Number SDIO 84-88-C-0018. The comments are welcome. Please send comments electronically to johnsons@jedi.sdio.mil or by mail to CPT Stephen L. Johnson, SDIO/ENA, The Pentagon, Room 1E149, Washington, D.C. 20301-7100.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

EXECUTIVE SUMMARY

This document is the Final Report on Task Order 14, Subtasks 3 and 4 on BM/C^3 Technology (Multi-Grain) performed under contract to SDIO. The prime contractor for this effort has been The Analytic Sciences Corporation, with SPARTA and Teledyne-Brown Engineering as sub-contractors. SPARTA was designated technical lead for this work.

Task Order 14 is concerned with Multigrain (that is, parallel) computer architectures in support of SDI BM/C^3 processing. It began in late April of 1988, and two subtasks were initially assigned, over a six-month period. The first of these subtasks was to conduct a survey of the current state-of-the-art in parallel processing architectures. In order to assess the applicability of these architectures, four representative BM/C^3 algorithms were selected and "mapped onto" each of the eight architectures in Subtask 2. That is, parallel algorithms appropriate to each architecture were developed, and performance estimates were derived based on instruction counts and data size estimates.

The area that appeared to be of most concern at the conclusion of this phase of Task Order 14 work was that the four representative BM/C^3 algorithms were analyzed in isolation. A more realistic and convincing analysis would assemble a complete suite of BM/C^3 algorithms, together with realistic time lines and data set sizes. In this way, the true size of a parallel architecture sufficient to perform the full processing task could be estimated with increased confidence.

An extension, reported on in this report was organized into two phases running sequentially for three months each -- Subtasks 3 and 4, respectively. Major deliverables included a design of a software tool (3-month point, the conclusion of Subtask 3), and the software and documentation (due at the end of Subtask 4). A second direction of the work dealt with constructing a test case of sufficient complexity to stress the tool. It was agreed that the test case would consist of mapping the AOA (Airborne Optical Adjunct) Tracking algorithm (the software component of the test case) onto an AOSP (Advanced On-board Signal Processor) hardware configuration (the hardware component of the testcase). A hand-worked example of a significant sub-case constituted the 3-month deliverable, with the complete test case and accompanying analysis provided at the 6-month point, the conclusion of Subtask 4.

This report provides the record of the work performed under Subtasks 3 and 4.

Section 1 is the System Description of PERM (Processor Ensemble Runtime Model). Section 2 is the PERM User's Guide.

PERM consists of three major components called, respectively, IPERM, CPERM, and DPERM. Briefly, IPERM allows the user to specify a model of both the system hardware (Processor Ensemble) and Software (Tasks). This can then be processed by CPERM to create files of summary statistics. These files can then be accessed by DPERM to display the data in tabular or graphical form.

To exercise the PERM model concepts, a tracking model example was implemented. The target software was the Track Model of the AOA Mission Data Processor. The target software was the Advanced On-Board Signal Processor (AOSP). Several assumptions were used. For example, only two of the six track nodes were modeled, to limit the testcase to a reasonable analysis effort while still exercising parallel processing features.

The track task was modeled in PERM as 13 threads divided into 46 application segments. Fourteen join segments representing dependencies were also included. The application was implemented in PERM constructs and run on the IBM PC/AT. Numerical results and graphical outputs with reasonable values resulted. Several important conclusions include:

- PERM is relatively user-friendly. Data entry, though somewhat tedious, is self-explanatory with the help of pop-up menus and scroll options.
- PERM provides relatively "quick" response. Computational runs are relatively fast, and graphs can be produced in seconds by reducing data with the selected options. Organization can also speed data entry, thus increasing turn-around rate.
- Like all simulation tools, PERM has limitations. One constraint, its memory, could be relaxed by employing more efficient software methods or by porting the software to a workstation.

PERM is meant to be used as a feasibility tool. When performing detailed analysis, PERM should be used with, not instead of, a higher fidelity simulation. The initial version of PERM represents a very early stage in the life-cycle of each a system. Many follow-on enhancement possibilities have been identified for possible future work.

0.0. INTRODUCTION

This document constitutes the Final Report on Task Order 14, Subtask 3 and 4, performed under contract to SDIO. The prime contractor for this effort has been The Analytic Sciences Corporation, with SPARTA and Teledyne-Brown Engineering as sub-contractors. SPARTA was designated technical lead for this work.

The following two sub-sections provide the background, and an overview of the contents of this report.

0.1 BACKGROUND

Task Order 14 is concerned with Multigrain (that is, parallel) computer architectures in support of SDI BM/C^3 processing. It began in late April of 1988, and two subtasks were initially assigned, over a six-month period. The first of these subtasks was to conduct a survey of the current state-of-the-art in parallel processing architectures. The survey was to be from the point of view of possible space-basing of these architectures. Thus, a number of issues were to be kept in mind, including: size/weight/power, radiation hardening, level of integration, security, fault tolerance, and reliability. The result of Subtask 1 was a briefing in which eight of the most promising architectures were selected for a more intensive examination of their applicability to BM/C^3 processing.

The second subtask took up at the point where the first left off. In order to assess the applicability of these architectures, four representative BM/C^3 algorithms were selected and "mapped onto" each of the eight architectures. That is, parallel algorithms appropriate to each architecture were developed, and performance estimates were derived based on instruction counts and data size estimates. In addition, detailed estimates of S/W/P were made for each of the architectures when placed in a space-qualified configuration, and an assessment was provided for security, fault tolerance, and reliability. The results of this study (and the subtask 1 materials) were presented in a Final Report delivered in September 1988.

The area that appeared to be of most concern regarding the initial phase of Task Order 14 work was that the four representative BM/C^3 algorithms were analyzed in isolation. A more realistic and convincing analysis would assemble a complete suite of BM/C^3 algorithms, together with realistic time lines and data set sizes. In this way, the true size of a parallel architecture sufficient to perform the full processing task could be estimated with increased confidence. An initial suggestion, then, was that an automated model be constructed for one of the architectures thought to be of greatest interest.

As the approach received wider review, however, it was observed that such a model, if intimately tied to a single particular architecture, would be of only limited utility. Better would be a tool that could model a variety of hardware architectures, and a variety of software algorithms running on those architectures.

The broad outlines of such a tool were sketched by Dr. Harold Camp of MIT Lincoln Laboratory. It was recognized that development of such a tool using the original schedule introduced considerable risk into the program. No additional funds or time were to be provided, even though the development effort was of a different and substantially broader scope than had been originally envisioned. However, it was felt by potential users of the tool (especially the POET) that only a tool with the added generality and flexibility would be of real use to the program. Further, Dr. Camp agreed to make some of his time and technical advice available, especially during to the initial part of the project, to help in the design effort and to validate the tool.

The program was organized into two phases running sequentially for three months each for Subtasks 3 and 4, respectively. A further separation was made between two directions of the work. The first was the development of the software tool itself. Major deliverables included a design (due of the 3-month point, the conclusion of Subtask 3), and the software and documentation (due to the end of Subtask 4). The second direction of the work dealt with constructing a test case of sufficient complexity to stress the tool. The testcase should also be of interest to SDIO, independent of the tool. It was agreed early in the project that the test case would consist of mapping the AOA (Airborne Optical Adjunct) Tracking algorithm (the software component of the testcase) onto an AOSP (Advanced On-board Signal Processor) hardware configuration (the hardware component of the testcase). A hand-worked example of a significant sub-case would constitute the 3-month deliverable, with the complete testcase and accompanying analysis provided at the 6-month point, the conclusion of Subtask 4.

0.2 OVERVIEW OF THE REPORT

This report provides the record of the work performed under subtasks 3 and 4. At the top level, it divides into documentation of the tool (Sections 1 and 2) and a description of the test case (Section 3).

Section 1 is the System Description. It begins with a top-down look at the tool, introducing and motivating the modeling concept and terminology. Then, detailed descriptions of the hardware and software modeling capabilities are provided, including PERM outputs, and a complete worked example. A listing of the example, as specified in PERM data structures, is also provided in Appendix A.

Section 2 is the PERM User's Guide. It describes the hardware configuration(s) necessary to support PERM, and provides a comprehensive command-by-command description of the PERM user interface. These are also available via the on-line Help capability of PERM.

Section 3 presents the testcase. The AOSP hardware model and the associated AOA software model are presented and explained. Then, performance data generated by PERM is analyzed. A discussion is also provided of the experience using PERM on a real problem -- both strengths and weaknesses.

The present version of PERM may be considered a prototype (although not a throwaway), at the earliest stage of life cycle. There is potential for numerous improvements. Code modules can be tightened and optimized. Memory management can be refined. Despite successful test cases, the possibility cannot be completely excluded of deep, logic "bugs". With practical field experience, enhancements and tidy-up opportunities may be identified and implemented.

1. SYSTEM DESCRIPTION

This Section of the Final Report will provide an overview of the PERM modeling tool. Sections are included that describe: the software design (Section 1.1); modeling a processor ensemble (Section 1.2); modeling software running on the processor ensemble (Section 1.3); the computational strategy PERM uses to collect performance statistics (Section 1.4); PERM outputs (Section 1.5); how PERM can be used to support the system design process (Section 1.6); and limitations that should be taken into account when considering its use (Section 1.7).

PERM will allow an experienced analyst to observe concurrent, asynchronous resource utilization by processes running on an array of processors. The resources in question are memories and busses. As a process runs on a processor, it makes demands on system memories and busses; concurrently, other processes running on other processors may also be making demands on these same resources. PERM provides a mechanism for modeling these resource demands and displaying, for each resource, the aggregate demand from all sources as a function of time. This means providing models both for the resources and their connectivity -- the system hardware -- and for the demands against those resources over time -- the software. It also means providing a computation engine to drive the resource utilization and to keep statistics. Finally, it means providing a data display capability so that the resource utilization data can be observed and analyzed.

The following sections describe how PERM provides these capabilities, and how the analyst would go about using PERM to solve problems of interest in system design and engineering. Section 2 then provides detailed information on running PERM and on its user interface.

1.1 OVERVIEW OF THE PERM SOFTWARE DESIGN

PERM consists of three major components called, respectively, IPERM, CPERM, and DPERM. Briefly, IPERM allows the user to specify a model of both the system hardware (Processor Ensemble) and Software (Tasks). This can then be processed by CPERM to create files of summary statistics. These files can then be accessed by DPERM to display the data in tabular or graphical form. This relationship is illustrated in Figure 1-1. The following sections describe the functions of these three modules in greater detail.

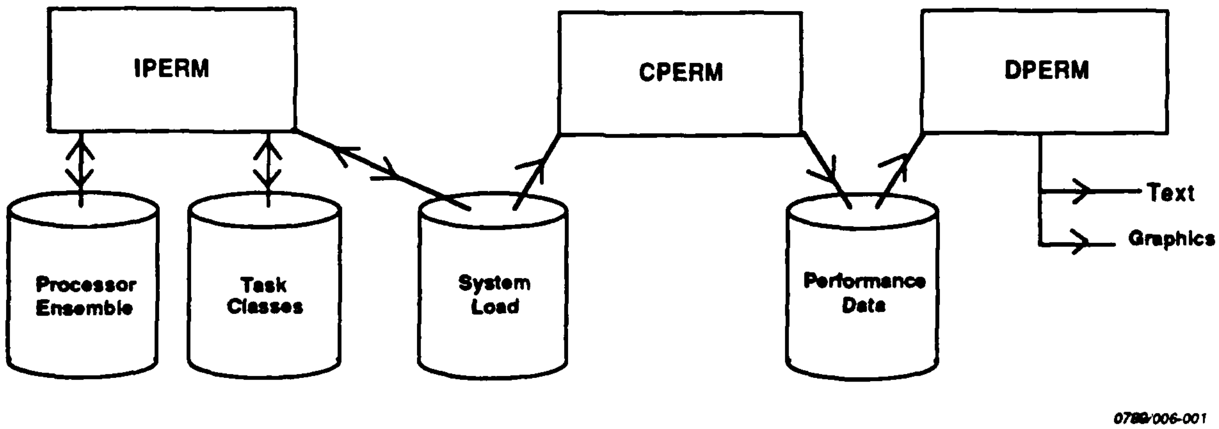


Figure 1-1 Major PERM Software Modules

1.1.1 IPERM

This is the largest of the three software components of PERM. Its purpose is to allow the user to specify a model of both the system hardware (the Processor Ensemble) and software (Tasks). The data structures built by IPERM are suitable for input to the next software module -- CPERM -- which performs the computational part of the work, and produces statistical performance data.

IPERM itself consists of three sub-modules: one for specifying a Processor Ensemble; one for specifying one or more software entities to run on the hardware (Task Class definition); and one to piece together a number of Task Classes into a System Load suitable for input to CPERM. Logically, these proceed in order -- first hardware, second software modules to run on the hardware, and third the complete sequence of software tasks to be processed. Thus, at the top level, IPERM provides three classes or types of operations: Processor Ensemble operations; Task Class operations; and System Load operations. This is illustrated by Figure 1-2.

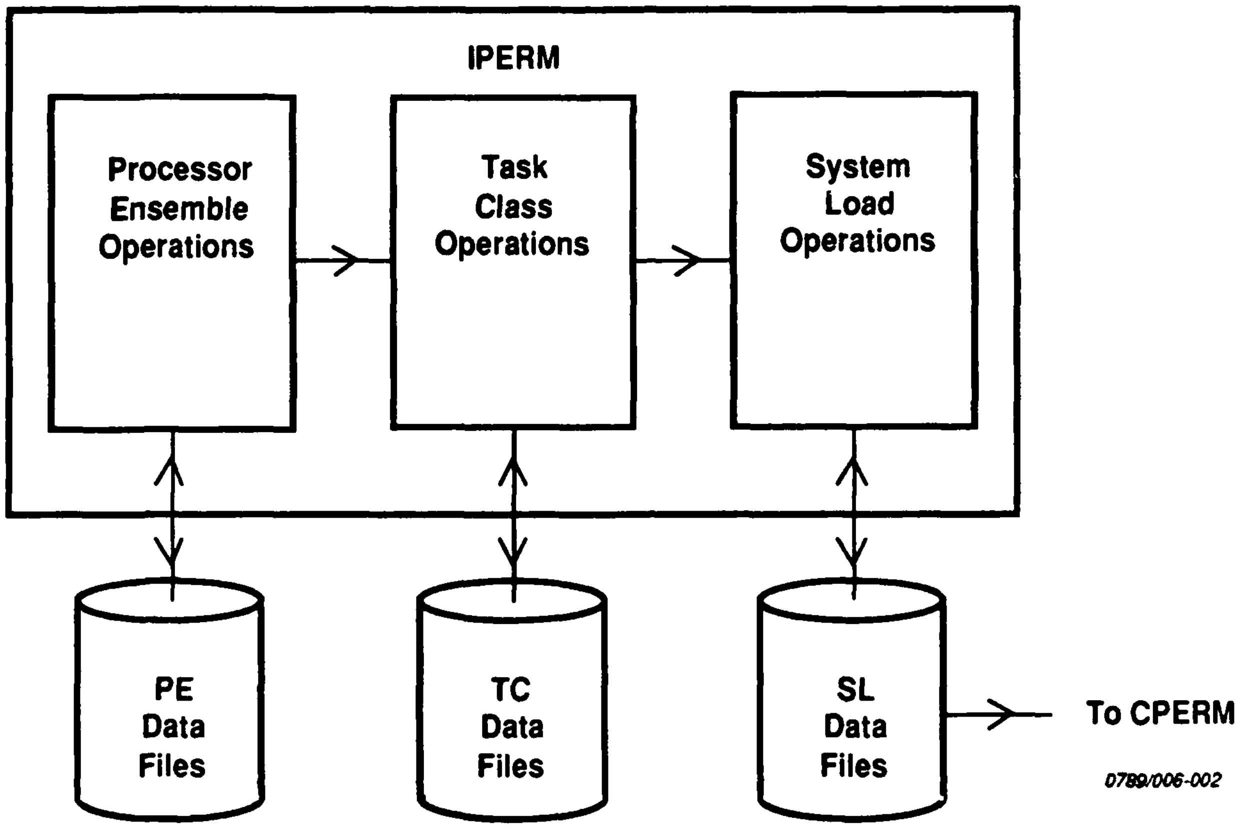


Figure 1-2 Sub-Modules of IPERM

Section 1.2 below will go into greater detail about how to model system hardware using the Processor Ensemble operations. Section 1.3 describes in greater detail about how to model system software and loads using Task Class and System Load operations.

1.1.2 CPERM

The purpose of this module is to generate resource utilization statistics based on a PERM model created using the IPERM modeling capabilities. The user interface is minimal: merely specify the name of the data file holding the System Load definition as created by IPERM. The software then "drives" the resources, keeping track of individual and aggregate demand by the processors for the memories and busses. The files produced by CPERM can then be accessed by DPERM, the third PERM software module -- for tabular and/or graphical display.

1.1.3 DPERM

Section 1.4 below provides a more detailed look at the computational algorithm used by CPERM. This module provides the ability to view and analyze the resource utilization statistics gathered by CPERM. Two displays provide an event-by-event record of all activity. Two others provide a resource-vs-time display so that, for example, the bus bandwidth requirements (for any bus in the PERM model) can be compared, as a function of time, to the actual bandwidth provided by the bus. Section 1.5 below describes in greater detail the particular kinds of graphical displays output by DPERM.

1.2 MODELING A PROCESSOR ENSEMBLE IN PERM

This section will describe the way in which an analyst would model a Processor Ensemble -- the underlying hardware resources (processor, memories, and busses) of the system. An initial section introduces the basic concepts and terminology. Then, the notion of Classes and Instantiations is taken up -- a key notion in the PERM modeling strategy. Finally, an example is provided which illustrates the notions of the previous sections in a concrete way.

1.2.1 Basic Concepts and Terminology

PERM recognizes three basic hardware entities: Processors, Memories, and Busses. Each will be described in turn.

In PERM, a Processor is conceived of as making demands on other system resources -- memories and busses. Any software that runs on the system always, in PERM, runs on a processor; and as it runs, it makes demands on the memories and busses connected to (that is, associated with) its processor. Thus, the primary property a processor has, in the PERM scheme, is its connectivity to other processors, memories and busses. That is, the defining property of a processor is the list of other processors, memories, and busses to which the processor is connected. In defining a processor to PERM, the analyst must therefore specify:

- the list of connected processors;
- the list of connected memories; and
- the list of connected busses.

These lists, which specify the processor's connectivity, completely specify the processor in the PERM model.

A memory in PERM has two properties in addition to its connectivity to processors (that is, the list of processors that can access it). Each memory has a size (in Bytes), and a bandwidth (in Bytes/Second). Software running on a processor can make demands for both these resources: for a certain amount of memory required, and for a certain amount of memory I/O (both Reads and Writes, normalized to Bytes regardless of the word size). All processors connected to a memory (that is, in the list of processors to which the memory is accessible) can, concurrently and asynchronously, make demands against both these resources. PERM keeps track of this utilization over time, and produces reports and graphics that display it for analysis. In defining a memory to PERM, the analyst must therefore specify:

- the list of processors that can access the memory;
- the size of the memory (in Bytes); and
- the I/O bandwidth of the memory (in Bytes/Second).

A bus in PERM has one property in addition to its connectivity to processors (that is, the list of processors that can access it). This is its bandwidth (in Bytes/Second). Software running on a processor can make demands for this resource -- that is, can utilize a certain amount of the bandwidth. Each processor attached to the bus can, concurrently and asynchronously, make such demands. PERM keeps track of this utilization over time, and produces reports and graphics that display it for analysis. In specifying a bus to PERM, therefore, the analyst must enter:

- the list of processors that can access the bus; and
- the bandwidth of the bus (in Bytes/Second).

In the next section, it will be shown how PERM enables the analyst to enter these lists and performance parameters in an orderly and efficient manner.

1.2.2 Classes and Instantiations

In defining processors, memories, and busses, PERM utilizes the notions of Classes and Instantiations. Conceptually, the Class defines generic properties which are inherited by all members of the class. An instance of the Class is specified both by its name and by its membership in the Class.

For memories, for example, the size and bandwidth parameters are entered once, and are inherited by all Class members. Similarly, the bandwidth parameter is entered once for a bus Class, and is then automatically assigned to all members of that Class.

However, as we have seen, a major defining property of PERM objects is their connectivity -- that is, the processors, memories, and busses associated with a particular system resource. Thus, a Class also has a connectivity pattern: each member of the Class has a specified number of other processor, memory, and bus Classes that are accessible to it.

Specifically, we recall that a processor is defined by three lists: the accessible processors; the accessible memories; and the accessible busses. Thus, to create a processor Class, it is necessary to create three generic lists. When a processor of the Class is instantiated, these lists can be filled in with names of other instantiated processors, memories and busses to which the specific processor is connected. These generic lists are created by assigning variable names as place-holders. For the generic list of accessible processors, for example, one variable is named for each accessible processor, and the Class of that processor is also specified. When an instance of the Processor Class is created, these variable names are then replaced by the names of instantiated processors of the designated Class. In this way, the generic lists that describe the Class are turned into real lists describing the actual connectivity of the Processor Ensemble.

In a similar way, a Processor Class will have a generic list of memories (and their Classes) designated by variable place-holders. Likewise a processor Class will have a generic list of busses (and their Classes) designated by variable place-holders.

An identical device is used for describing the connectivity pattern for memory and bus Classes. For these, however, it is only necessary to specify the accessible processors: memories are not connected to memories or busses; and busses are not connected to memories or busses (at least, not within the PERM modeling strategy). When a memory Class is created, variable place-holders are entered (with an associated processor Class) for each processor that can be connected to the memory, and similarly for bus Classes. When a memory (or bus) is instantiated, these variables are replaced by the names of instantiated processors (of the associated Class).

1.2.3 Example of a Processor Ensemble

We will now illustrate the notion of Classes and Instantiations with an example. Figure 1-3 shows a simple hardware configuration consisting of two identical processor boards connected to each other and to a global memory by a system bus. Each processor board, in turn, has three components: a central processing unit, a local memory, and a direct-memory-access controller (so that the CPU can proceed to process in parallel with memory and bus I/O).

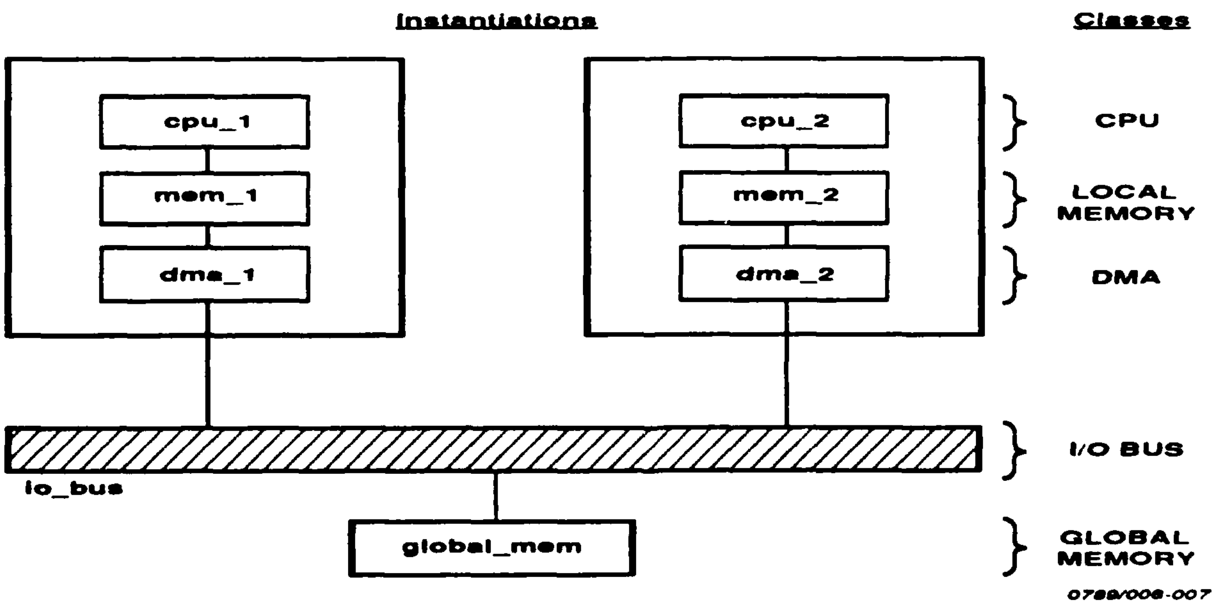


Figure 1-3 Example Processor Ensemble

In PERM, we would model this using:

- two processor classes, one for the CPU, and one for the DMA. Each Class will have two members.
- two memory classes, for the local memories, and one for the global memory. The local memory class will have two members, and the global memory class will have one member.
- one bus class, with one member, modeling the system bus.

If we look, for example, at the processor Class associated with the CPUs, we see the following connectivity pattern: each CPU is accessible to one local memory and to one DMA. Thus, we will have one variable place holder of class DMA, and one variable place holder of class LOCAL MEMORY. Let us give the DMA variable place-holder the name "v_local_dma", and the LOCAL MEMORY variable place-holder the name "v_local_memory". Then the CPU Class definition is given by:

variable name	Class
v_local_dma	DMA
v_local_memory	LOCAL MEMORY

In a similar way, we can define the DMA class:

variable name	Class
v_local_cpu	CPU
v_other_dma	DMA
v_local_memory	LOCAL MEMORY
v_global_memory	GLOBAL MEMORY
v_system_bus	I/O BUS

The definition for the memory Class LOCAL MEMORY:

variable name	Class
v_local_cpu	CPU
v_local_dma	DMA

The definition for the memory Class GLOBAL MEMORY

variable name	Class
v_first_dma	DMA
v_second_dma	DMA

The definition for the bus Class I/O BUS

variable name	Class
v_first_dma	DMA
v_second_dma	DMA

Once the Classes have been defined, instances of the Classes can be created. Class CPU has two instances: cpu_1 and cpu_2. Class DMA also has two instances: dma_1 and dma_2. Memory Class LOCAL MEMORY has two instances: mem_1 and mem_2. Memory Class GLOBAL MEMORY has only one instance: glob_mem. And finally, the bus Class I/O BUS has one instance: io_bus.

For each instance, we can assign to each variable in the Class definition a corresponding instance from the associated Class. For example, consider the instantiation of "dma_2" from processor class DMA. We would have:

variable name	instance from	Class
v_local_cpu	cpu_2	CPU
v_other_dma	dma_1	DMA
v_local_memory	mem_2	LOCAL MEMORY
v_global_memory	glob_mem	GLOBAL MEMORY
v_system_bus	io_bus	I/O BUS

The reader will find a complete listing of this example Processor Ensemble in Appendix A, including all Class definitions and instantiations.

It will be observed that naming conventions are a great help in keeping track of the PERM entities and their inter-relationships. This is up to the user, and in fact most of the modeling work must be done in advance of sitting down at the terminal. The PERM interface merely provides an orderly way of entering the description of the conceptual model.

As we noted, Memory classes also have two additional parameters: size (in Bytes), and I/O bandwidth (in Bytes/Second). Likewise, a Bus Class also has a parameter: bandwidth (in Bytes/Second). These parameter values for the Classes in our example are as follows:

LOCAL MEMORY:

Size: 4,000,000 Bytes

Bandwidth: 30,000,000 Bytes/Second

GLOBAL MEMORY:

Size: 60,000,000 Bytes

Bandwidth: 10,000,000 Bytes/Second

I/O BUS:

Bandwidth: 15,000,000 Bytes/Second

NOTES:

1. There is another bus, not modeled in the example presented above: the bus connecting the local DMA and CPU to the local memory. This bus, however, is of a quite different sort than the packetized system bus which is in the model. The point of including a resource -- bus or memory -- in a PERM *model is interesting in how that shared resource* is used. In this case, the "bandwidth" of the local bus (if that terminology makes sense in this context) is not an issue. The memory bandwidth, however, is.

2. It may seem at first unusual to include the DMAs as separate objects in the PERM model. The motivation for this is to model the concurrent use of the local memory and global memory. For example, when a data transfer between the local memory and global memory is being made, some portion of the bandwidth available from those resources is used by the DMAs. This leaves correspondingly less bandwidth available to the CPUs, and may result in a reduction of their processing rate (due to idle time waiting for access to the memory). It is exactly this kind of phenomenon that PERM is intended to capture.

3. The "accessible processor" list associated with a processor Class -- that is, the processors with which a particular processor can synchronize -- does not currently have any role in PERM processing. If desired, it can be omitted entirely from a PERM model without affecting in any way the results. For PERM, the key notion is the connectivity between the processors, on the one hand, and the resources they access (busses and memories), on the other.

1.3 MODELING SOFTWARE IN PERM

In this section, we will examine how an analyst can construct a model of the software suite to be executed on a Processor Ensemble. We will begin with an introductory overview that presents the underlying concepts and terminology. This is followed by individual sections explaining in greater detail the semantics of the software model objects that PERM recognizes and manipulates. Finally, the example begun in Section 1.2.3 above is continued.

1.3.1 Basic Concepts and Terminology

In this section, we will give a top-down overview of how PERM models software running on a Processor Ensemble. Subsequent sections will then present a bottom-up look at each of the objects in greater detail.

At the top level, PERM views the software as an acrylic directed graph of Tasks, called a System Load. A Task "runs" on a group of processors -- a subset of the full array of processors described in a Processor Ensemble. It may consist of the full array, or it may consist of a subset of the array. Tasks can run concurrently -- that is, they can simultaneously be active in the model, concurrently making demands on any resources (busses or memories) shared by processors assigned to the Tasks.

An example is shown in Figure 1-4. The System Load shown there consists of five Tasks. In this example, the first two (top-most) tasks can, if they do not share any processors in common, execute concurrently. Task 3, however, cannot begin until both Tasks 1 and 2 have completed. Task 4 could, in theory, run concurrently with Tasks 1 and 2. Its position in the diagram, however, suggests that its start may be delayed, either because it requires some processors initially assigned to Task 1 and 2, or because it has a delayed start time (its data is not available until after an initial delay). Similarly, Task 5 cannot begin until Tasks 3 and 4 have completed.

In addition to a start time dependency (due to completion of previous Tasks, availability of resources, or initial start time delay), a Task has an input data set size. In Figure 1-4, these are indicated by the symbols N1, N2, .. , N5. The role of this parameter will be explained shortly. Suffice to say, at present, that the data size parameter is associated with a Task (and hence, with all objects comprised by the Task).

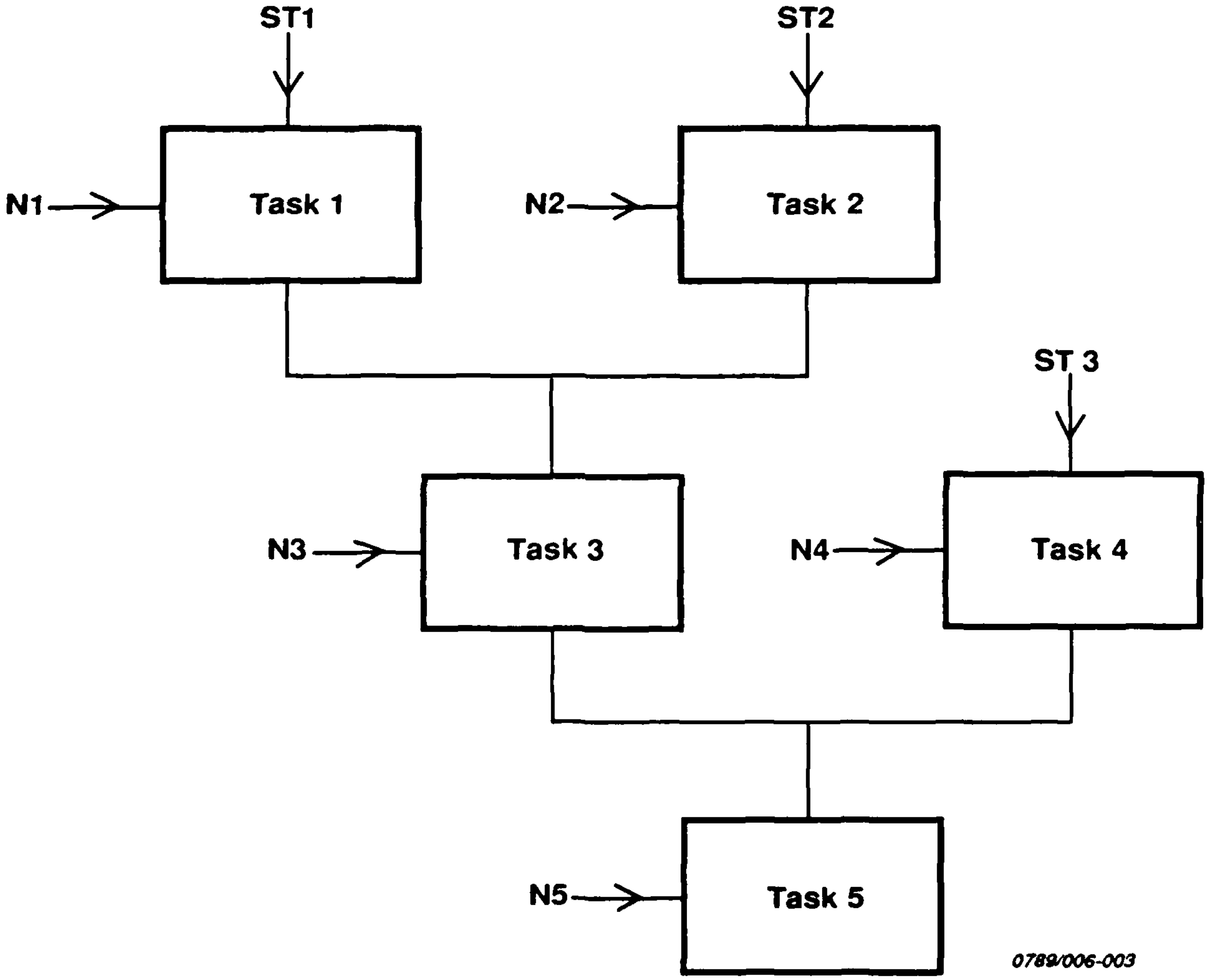


Figure 1-4 An Example of a System Load

Figure 1-5 shows the interior of a Task. Each of the vertical sequences of blocks is called a Thread, and each Thread is associated with a particular instantiated processor in the Processor Ensemble. The blocks on the Thread are called Segments (see below), and are used to model the system resource utilization during the time when the segment is active. The Thread, thus, specifies the sequential order in which the segments are to "execute". In addition, just as the start of a Task can be dependent on the completion of another Task or Tasks, so can the start of a

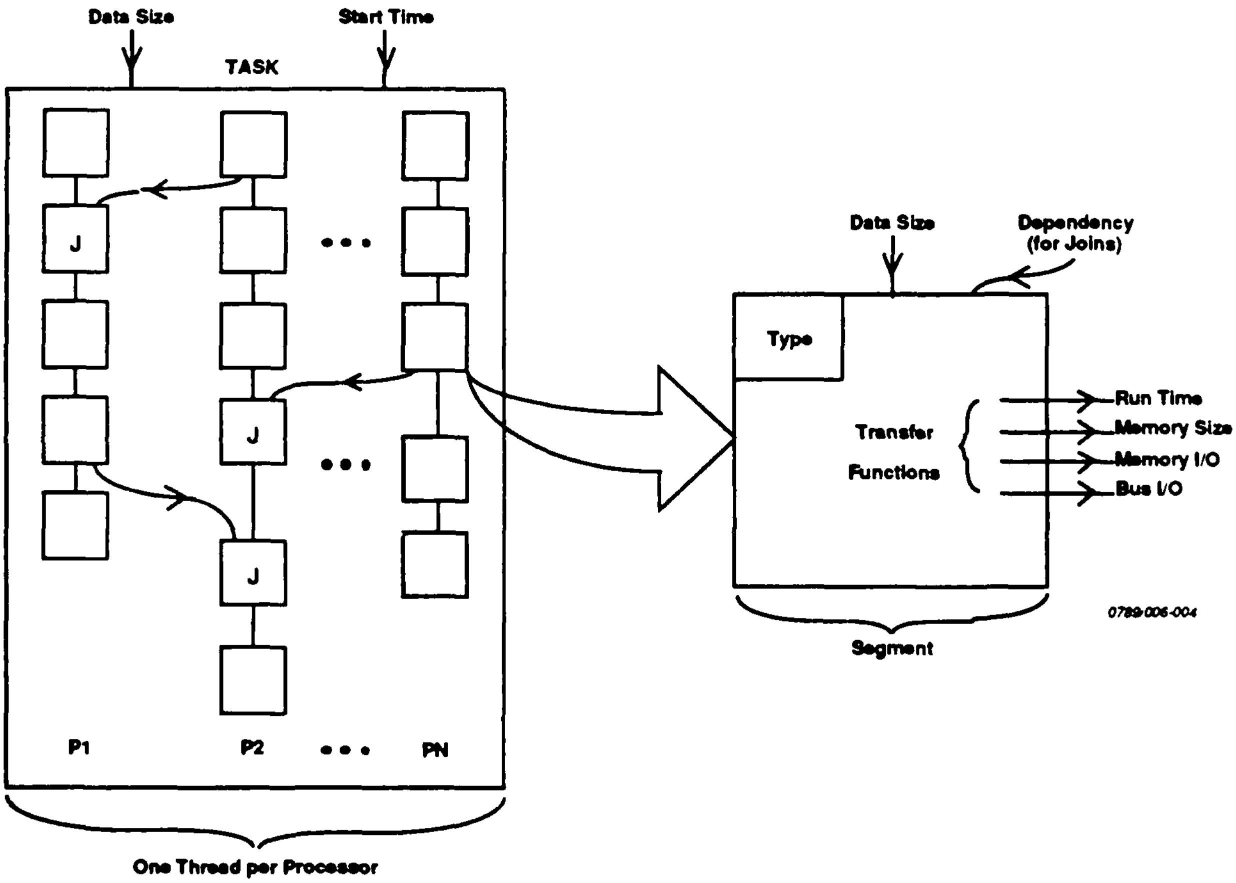


Figure 1-5 Threads and Segments

segment be conditioned on the completion of another segment in the Task -- both its immediate predecessor in the Thread, and also some other segment on a different Thread. PERM provides a natural way to build up sequences of segments into Threads, and to specify any inter-thread execution dependencies that may exist.

If we now look inside a Segment (Figure 1-5), we will find a number of model parameters that are used to characterize the way in which the Segment will behave when its turn to execute arrives. The most important of these are Transfer Functions (see Section 1.3.2.1 below). Transfer Functions are the means by which PERM specifies resource utilization -- run time, memory size and bandwidth, and bus bandwidth. Basically, the idea is as follows. A segment belongs to a Thread, and a Thread belongs to (that is, "runs on") an instantiated processor. That processor, in turn, can access system resources -- whatever busses and memories are accessible to it in the Processor Ensemble. For each of these resources, a Segment will specify (by means of Transfer Functions) the amount of that resource -- memory size and bandwidth for memories, and bus bandwidth for busses -- required during its period of execution. The reason they are called Transfer Functions (as opposed, say, to constants) is that they accept the Data Size parameter as input, and compute the resource utilization and run time as an output; different sizes of data input result in different values of run time and resource utilization. The exact form of a Transfer Function, and how the analyst specifies it, is discussed in greater detail in Section 1.3.2.1 below.

In addition to its parent Thread and its Transfer Functions, a Segment has two other characteristics. One is its type -- Application Code, Operating System, or Join -- which are used for accounting purposes in the Compute phase of PERM. The other is the execution dependency -- that is, for a Join Segment, the Thread/Segment whose completion is required before execution can continue. Finally, just as a Task can have an initial time offset, so can a Join Segment, so that an entire Thread, or part of it, can be delayed by some fixed, known amount if that is desired.

1.3.2 Segments

We now begin a bottom-up discussion of the objects that have been introduced above. We begin by discussing Transfer Functions and how the analyst derives and specifies them. Then the notion of Classes and instantiations is presented. Finally, the special role of Join segments in specifying execution dependencies and time offsets is explained.

1.3.2.1 Transfer Functions – As explained above, a Segment "runs" on a processor; and the processor in turn can access a specified set of system resources -- busses and memories -- as indicated in the Processor Ensemble when the processor was instantiated. The purpose of the Transfer Functions is to determine how much of each of these resources (and run time) the Segment requires. In addition, it is desired that the resource utilization be a function of data size: the larger the data size, the longer the run time, and the greater the resources required.

The exact form of a Transfer Function is given as follows. The analyst specifies (inputs) the values of six parameters:

R (the reduction factor)
Q1, Q2, and Q3 (the quadratic coefficients)
L1 and L2 (the log-linear coefficients)

The transfer function then computes the value, V, as a function of data input size, N, as follows:

$X = R * N$, and
$V = Q1 + Q2 * X + Q3 * X^2$
$+ (L1 + L2 * X) * LOG2(X)$
where V is the desired output as a function of N.

Thus, by choosing the six parameters R, Q1, Q2, Q3, L1 and L2 appropriately, the analyst specifies the output value V of the Transfer Function for any input value of data size N.

These parameters must, of course, be specified separately for every Transfer Function associated with a Segment. The number of such Transfer Functions is determined by the accessible resources. Every Segment has a Run Time Transfer Function. It also has two Transfer Functions for each accessible memory -- one to specify the amount of memory (in Bytes), and the other to specify the number of Bytes of I/O (including data and instructions, both Reads and Writes). The number of Bytes of I/O can be divided by the Run Time to compute Memory Bandwidth. Finally, there will be one Transfer Function for each accessible bus, to specify the number of Bytes to be sent on the bus during Segment execution. This number can then be divided by the Run Time to obtain bus bandwidth.

Note that the bandwidth, both for memory and for busses, is an average over the life of the segment. If greater detail is required, the Segment should be broken up into smaller pieces -- that is, the single Segment should be replaced by two or more Segments that model individual parts of the Segment activity.

If we turn to the example Processor Ensemble from Section 1.2.3 above, we find that any Segment associated with a processor of Class CPU will have three Transfer Functions: one for Run Time, one for memory size in the LOCAL MEMORY, and one for memory I/O in the LOCAL MEMORY. Similarly, a Segment running on a DMA processor will have six Transfer Functions: one for Run Time, one for memory size on the LOCAL MEMORY, one for memory size on the GLOBAL MEMORY, one for memory I/O on the LOCAL MEMORY, one for memory I/O on the GLOBAL MEMORY, and one for bus I/O on the I/O BUS. We also see that the variable names given to these accessible entities (by Class) serve nicely to indicate the type of Transfer Function, and associated resource, to be specified. This will be discussed in greater detail in the next Section.

During the Compute phase, PERM will generate a complete record of system activity -- the start and end time of each Segment on every Thread and Task, as well as the resources utilized while each Segment is active. At any time, then, and for any resource, it is possible to determine which Segments are active at that time, and what their utilization of that resource is. PERM can thus generate a graph that shows, for any resource, its utilization as a function of time. This is the primary output of PERM. The pivotal role of the Transfer Functions thus becomes apparent.

It is up to the analyst to specify the Transfer Functions, and to ensure that the Threads and sequence of Segments that constitute these Threads do, indeed, model the behavior of interest. The analyst has complete freedom to include as much or as little detail as may be appropriate. PERM does not derive Transfer Function coefficients; it merely records them as entered by the analyst, and then displays the system behavior (derived from these Transfer Functions) as output.

1.3.2.2 Classes and Instantiations – Just as the notions of Class and Instantiation were used in specifying the Processor Ensemble, the same approach is used to specify the Segments that make up the Threads and Tasks. In particular, it is desirable that identical Segments (as given by their Transfer Functions) be allowed to appear on different Threads, or more than once on the same Thread. The Transfer Functions should be specified once, and then inherited by every instance of the Segment. Thus, if the analyst edits the Segment definition (say, to change a Transfer Function coefficient), that change will be immediately inherited by all instances of that Segment.

This is accomplished by the notion of a Segment Class. A Segment Class is specified by the following information:

- the Type of the Segment -- Operating System, Application Code, or Join
- the Processor Class associated with the Segment, taken from the Processor Ensemble definition
- the coefficients of the Run Time Transfer Function
- for each accessible memory, as indicated by the list of variables (and their Classes) in the Processor Class definition, the coefficients for the memory size requirements Transfer Function (in Bytes)
- for each accessible memory, as indicated by the list of variables (and their Classes) in the Processor Class definition, the coefficients for the memory I/O Transfer Function (in bytes)
- for each accessible bus, as indicated by the list of variables (and their Classes) in the Processor Class definition, the coefficients for the bus I/O Transfer Function (in Bytes)

By specifying the Transfer Functions for the variables in the Processor Class, it is possible for PERM to then associate those Transfer Functions using the instantiated values of those connectivity variables when the Segment Class is associated with a particular instantiated processor. Thus, for example (referring to Section 1.2.3), a processor of Class CPU has a connectivity variable v_local_mem associated with a memory of Class LOCAL MEMORY. When a Segment is defined to run on this Class of Processor, two Transfer Functions (one for size and one for I/O) will be defined for the variable v_local_mem. When this Segment is then associated with a particular instance of CPU -- say, with processor cpu_2 -- the Transfer Functions will then refer to the particular memory associated with the variable; in this case, v_local_mem for cpu_2 is assigned to mem_2. Thus, the generic connectivity structure used to define a processor Class is used again, for Segments, to define a generic set of Transfer Functions. These are then inherited whenever the Segment Class is instantiated.

The way in which Segment Classes are instantiated to form Threads and Task Classes is discussed at greater length in Sections 1.3.3 and 1.3.4 below.

1.3.2.3 Join Segments – The role of Join Segments in PERM requires separate discussion. A Join segment serves two purposes. First, and most important, it provides the means for specifying inter-thread sequential execution dependencies. For example, suppose a Segment on one processor requires data produced by a Segment on another processor before it can proceed. If the data is ready (that is, if the second Segment has completed), the first Segment can proceed without delay. However, if the second Segment has not completed, the first Segment must wait, and incur some idle time as a consequence.

In PERM, this means that the execution of the first Segment is conditioned on the completion of the second. The way PERM implements this is to insert a Join Segment immediately preceding the first Segment. When the Join Segment is added to the Thread, the analyst will be asked to specify the dependency -- in our example here, the identity of the second Segment. If, when the Join Segment is encountered, the second Segment has already completed, then execution immediately transfers to the first Segment (immediately following the Join Segment). If, on the other hand, the second Segment has not completed, then the Join Segment is active, and continues active until the second Segment completes (at which point the Join Segment terminates, and its successor, the first Segment, can begin).

The second use for a Join Segment is to delay the execution of a portion of a Thread for some fixed amount of time. That is, a time offset can be specified (in seconds, relative to the start time of the Task), and the Join Segment does not complete (that is, its successor in the Thread does not begin) until that time is reached.

No resource utilization is associated with Join Segments. That is, they do not have any Transfer Functions. Their Run Time, if any, is derived based on the relative completion times of their two predecessors -- the immediate predecessor in the Thread, and the "remote" predecessor on a different Thread. PERM accounts for time spent in Join Segments as idle time (see Section 1.5 below).

1.3.3 Threads

The usual course of events in building up to a complete description of the software is to begin by creating or importing a "pool" of Segment Classes. As described above, these Segment Classes are associated with Processor Classes (from the Processor Ensemble). Once the full collection of Segment Classes has been specified, the next step is to specify Threads made up of Segments belonging to one or another of these Segment Classes.

To create a Thread, the analyst must specify which instantiated processor, from the Processor Ensemble, the Thread is to "run" on. Then, the list of Segments that make up the Thread is drawn from the pool of Segment Classes. Note that the Processor Class of the Segment must agree with the Processor Class of the instantiated processor associated with the Thread. Since the Thread is a list, PERM provides a full set of "list manipulation" capabilities: append, delete, insert, etc. The result is an ordered sequence of Segments.

Whenever a Join Segment is added to a Thread, its predecessor Segment must be specified and, in fact, must already exist. This implicitly imposes an order on the sequence in which the various Threads and Segments are instantiated, and the analyst may need to alternate back and forth between Threads, adding a few segments here and there as successive dependencies are encountered.

We mention here two additional capabilities provided by PERM to ease the task of building Threads. First, Segment Classes can be copied from one Task to another. Thus, a "library" of Segment Class definitions can be built and shared between Tasks easily. Second, it frequently happens that two or more processors share what amounts to an identical Thread -- that is, the sequence of Segment Classes for the two Threads are the same. PERM permits the segments from one Thread to be copied to another wholesale. Note, however, that the dependencies in the Join Segments may need to be adjusted for the new instance.

1.3.4 Task Classes

We have spoken above of Tasks being composed of Threads which are, in turn, composed of Segments. This is not strictly correct. Rather, Threads and their Segments (and Segment Classes) belong to Task Classes. What is generic about a Task Class (as opposed to an instantiated Task) is the predecessor and successor Task dependencies. Recall, from Section 1.1, that a full System Load is made up to a directed graph of Tasks (together with data size specifications for each, and offset start times, if appropriate). However, we would like for identical copies of the same Task to appear, if desired, multiple times in the System Load. To accommodate this, we specify everything about the Task except its connectivity -- that is, its predecessors and successors in the directed graph. Instead, when the Task Class is created, variable place-holders are specified to indicate the to-be-created connectivity. During the definition of a System Load, these place-holders can be replaced by the names of real, instantiated Tasks. In this way, the directed graph is built in exactly the same way that the connectivity pattern for a Processor Ensemble was built using Classes and Instantiations.

Thus, the major work in building a Task Class is in specifying the Segment Classes (Transfer Functions) and then in building Threads. In System Load definition (see Section 1.3.5 below), these Task Classes are then "strung together" into a directed graph of Tasks that constitute the System Load.

1.3.5 System Load

We have finally returned to the top level again. At this stage in building a complete PERM model, the analyst has specified a complete Processor Ensemble and one or more Task Classes. It is now time to specify the System Load -- a directed graph of Tasks for input to the Compute phase of PERM. To specify the directed graph, each node must have the following information:

- the name of the Task
- the Task Class to which the Task belongs
- the Predecessors of the Task in the graph
- the Successors of the Task in the graph
- the offset time, if any, for Task initiation
- the size of the data set to be processed by the Segments in the Task

PERM provides facilities to enter and edit all these parameters. In particular, the forward and backward dependencies are specified by means of the variable place-holders created for each Task Class. Specifying the Class of the Task thereby specifies a set of forward and backward dependency variables whose values (other instantiated Tasks in the System Load) are to be filled in.

A complete System Load is the final product of IPERM. A copy is stored to disk, and is the input for CPERM (see Section 1.4 below).

1.3.6 Summary

We have now seen how a complete PERM hardware and software model can be constructed. The basic steps are as follows:

- 1.0 Build a Processor Ensemble by
  - 1.1 Constructing Classes of
    - 1.1.1 Processors
    - 1.1.2 Memories
    - 1.1.3 Busses
  - 1.2 and then Instantiating the Classes
- 2.0 Build one or more Task Classes by
  - 2.1 Constructing a Number of Segment Classes using Transfer Functions for
    - 2.1.1 Run Time
    - 2.1.2 Memory Size Requirements
    - 2.1.3 Memory I/O Requirements
    - 2.1.4 Bus I/O Requirements
  - 2.2 and then Constructing Threads from these Segment Classes
- 3.0 Build a System Load by Instantiating the Task Classes

These steps correspond exactly to the three major IPERM sub-components: Processor Ensemble operations, Task Class operations, and System Load operations. Note that Step 2 cannot proceed without the output from Step 1; and Step 3 cannot proceed without the output from Step 2.

This discussion leaves untouched a number of features and properties of PERM having to do with the user interface and the means provided to name the entities and define their properties. A complete discussion is provided in the User Guide (Section 2), and particularly the Help Files provided for every PERM command. Here is a short list of some of the most important features.

1. A menu-driven user interface keyed directly to the PERM data structures as described above.

2. Verification routines to validate the internal consistency of PERM data structures created by the analyst, with error messages pointing to problem areas.

3. Save and Load capabilities so that intermediate versions of the data structures can be stored on permanent media for later recall and editing.

4. Print and Display routines so that the current state of the data structures can be viewed and, if necessary, edited.

1.3.7 Example

Appendix A contains a complete listing of a PERM data structure -- Processor Ensemble, Task Classes, and System Load. The Processor Ensemble is that presented in Section 1.2.3 above. Here, we will discuss the Task Class and System Load portions of the example.

Three Task Classes have been defined, named (respectively) TRACKING, EPHEMERIS GENERATION, and BIG SORT/SEARCH. We do not intend to go into details about the semantics of the underlying software; indeed, for our purposes, we might just as well have labeled the Task Classes X, Y, and Z. Figure 1-6 shows the Thread list for the TRACKING Task Class, which uses both CPUs and both DMAs. The Segment naming convention (which is not a PERM requirement) names the segment by Thread Number and Segment Number within the Thread. Join and Operating System Segments have been labeled, and the Join dependency structure is indicated by arrows passing from one Thread/Segment to another.

Refer to the Appendix for the Segment Class of each Segment, and to the Segment Class for the Transfer Function coefficients. Diagrams similar to Figure 1-6 can be constructed for both the other two Task Classes. EPHEMERIS GENERATION uses only cpu_1 and dma_1; BIG SORT/SEARCH uses only cpu_2 and dma_2.

In Figure 1-7, we present the System Load. This consists of four Tasks: the first and last are both instances of the TRACKING Task Class, but have different data size input drivers (2,000 Tracks and 2500 Tracks, respectively). The other two Tasks run concurrently -- one on cpu_1 and dma_1; and the other in cpu_2 and dma_2. Again, refer to Appendix A for a complete listing of the System Load specification.

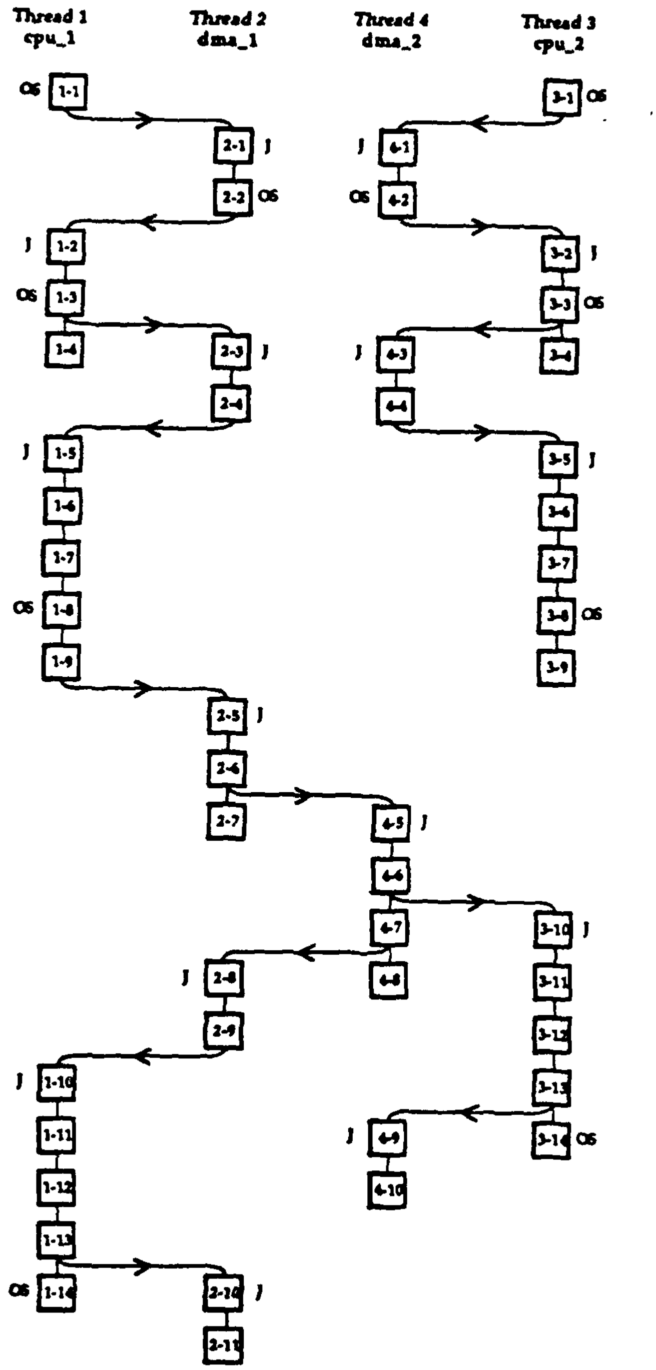


Figure 1-6. Threads and Segments for Tracking Task Class

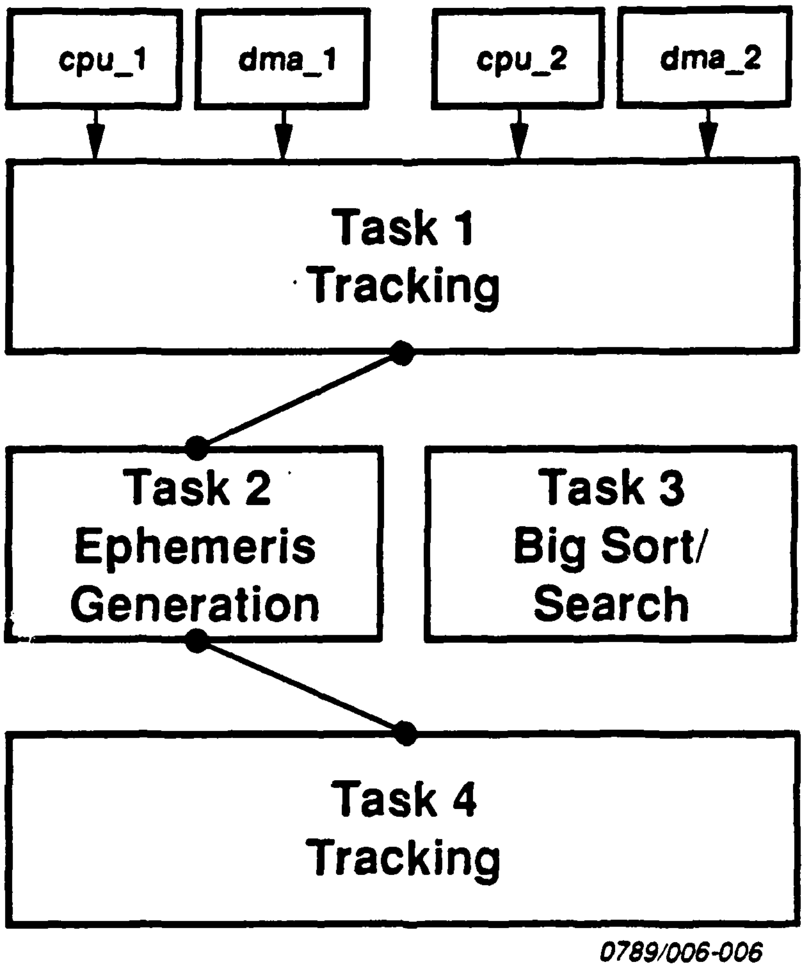


Figure 1-7 System Load Task Graph

There are a number of interesting features in this example which show how PERM is used to create good models of the system behavior. First, consider the collection of Segments 2-6, 2-7, 4-5, and 4-6. This sequence models the movement of a data set from the cpu_1/dma_1 complex to the cpu_2/dma_2 complex, using the system bus as the means. The Segment 2-6 is called the Send Data Lag, and reflects the initial time to notify dma_2 that data is on the way. The Join segment 4-5 is then satisfied, and Segment 4-6 kicks off. Thus, Segments 2-7 (Sending Data) and 4-6 (Receiving Data) operate concurrently, due to the presence of the short Lag Segment, 2-6. Without it, Segment 4-6 would wait until 2-7 was complete, not an accurate picture of what is actually occurring. As similar configuration occurs among the Segments 4-7, 4-8, 2-8 and 2-9, as the corresponding data moves back across the bus from cpu_2 to cpu_1.

Another instance worthy of note is the short time delay that has been put into Task 3, an instantiation of the BIG SORT/SEARCH Task Class. As it sits in the System Load, this Task has no forward or backward dependencies. If its Start Time were 0.0 (that is, if it were the same as Task 1), then PERM would arbitrarily decide which one to run first. This decision would be required, since Tasks 1 and 3 share resources (cpu_2 and dma_2). It order to force PERM to choose the right one (Task 1), we delay the start of Task 3 by a small amount (one millisecond in this example). In fact, of course, this results in its delay until all its resources have been released by Task 1.

Finally, note the first few Segments from Threads 1 and 2 in Task 1. These are intended to simulate the loading and initiating of a program from Global into Local Memory. The Segment Class names (Initialize, Load Instructions, Fork Process, Load Data) are semantically indicative of the function each is to perform.

1.4 COMPUTATIONAL STRATEGY

Once a complete and validated System Load has been built and stored to disk, the second major PERM module -- CPERM -- can be executed. There is not much for the user to do here. The name of the file holding the System Load definition must be given, and the name of the file(s) which will hold the output must likewise be specified. Otherwise, CPERM is simply computation.

As we have seen, every Segment has associated with it Transfer Functions that specify not only the Run Time of the Segment but also its utilization of any resources (memories and busses) accessible to the processor on which the Segment is "running". What CPERM does is to write a record corresponding to the start of each Segment. This record contains such information as the name of the Segment and processor, its duration (Run Time), and resource utilization values derived from the Transfer Functions. The input data size required by the Transfer Function is taken from the Task data size specified at System Load. Thus, all Segments within a Task will use the same data size value.

At each step, CPERM finds the next earliest Segment start time, scanning over all Segments in all Tasks. It then writes the corresponding entry to a file, called the Event History File, in PERM format. One of the outputs produced by DPERM is an ASCII version of this file. Except for Join completion events, which are handled separately, the file is time ordered; each entry is no earlier than the preceding one. Thus, a single pass through this file is enough for DPERM to generate any of its outputs, as we shall see.

Of course, this is something of a simplification, since delays due to Join segments must be taken into account, as well as lost time while one Task waits for another to complete. However, the basic idea is here: a file that makes a complete listing of every significant event in the run history of the System Load.

1.5 PERM OUTPUTS

Once the Compute phase of PERM has processed a System Load, the data generated can be displayed using the third major PERM module, DPERM. DPERM requires the name of the output files generated by CPERM.

There are four basic outputs by DPERM. We will discuss each in turn. However, they are all based on the Event History File produced by CPERM, as described in the preceding section. The first type of output is an ASCII version of the Event History File. That is, each record -- recording the start time and duration of each Segment, as well as its resource utilization -- is printed in ASCII form either to the screen or to a user-selected DOS file.

The second type of output is the same as the first, except that it relates only to the Segments associated with a single user-selected processor. Again, the data is either displayed at the screen or is sent to a DOS file. This output also produces usage statistics for the processor: the amount of time (and %) it spent in Operating System code and in Application Code. The remaining time is idle time, and is accounted for both by Join Segments and by starting and ending Tasks.

The third type of output concerns resource utilization. Recall that memories have two types of utilization: capacity, and I/O bandwidth. Busses have one type: bandwidth. For any resource, and any type of utilization, PERM will display the utilization of that resource (as a percentage of total available) as a function of time.

Figure 1-8 shows such a plot for bandwidth utilization of the LOCAL MEMORY mem_1 from our example. Both cpu_1 and dma_1 can access this memory, and the plot is of the sum of their utilizations. Only the time interval 2.0 to 6.0 has been selected. DPERM will also print out the actual values used to make the plot in ASCII form, either to the screen, or to a user selected DOS file.

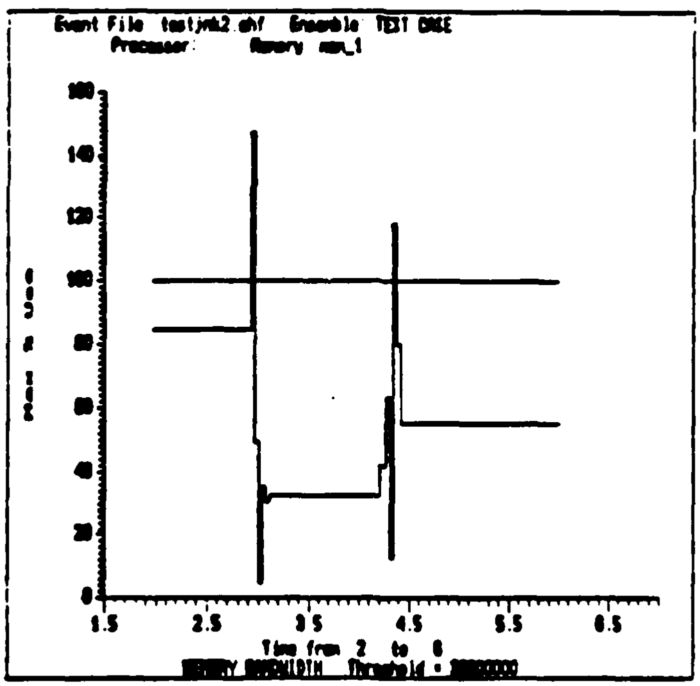


Figure 1-8 Total Resource Utilization

The fourth type of output is the same as the third, except that, instead of total utilization, only the utilization by a selected processor is displayed. Figure 1-9 shows the plot for the usage of mem_1 bandwidth by cpu_1 as a function of time. The same time interval was used for both Figures 1-8 and 1-9.

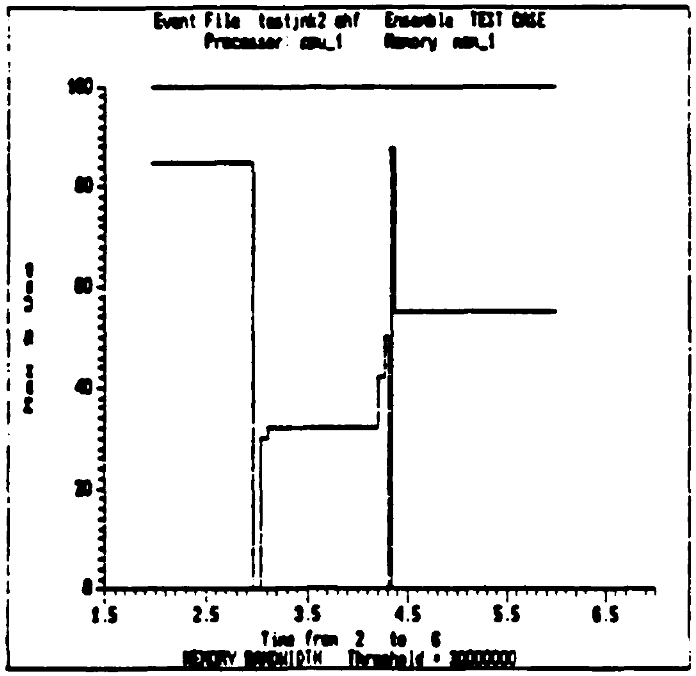


Figure 1-9 Processor/Resource Utilization

1.6 HOW PERM SUPPORTS THE SYSTEM DESIGN PROCESS

The major goals directing the design of PERM were the following:

- low cost, as reflected by required hardware and by analyst modeling time;
- medium fidelity: better than back-of-the-envelope, but not as good as low level event-driven simulations; and
- rapid turn-around.

PERM is ideally suited to support feasibility analyses. That is, it compares the aggregate resource utilization against the available resources, and reports. If the report is not acceptable (that is, for example, if the total memory required exceeds the total available), PERM cannot fix the problem. Fixing the problem is up to the analyst, either by providing more memory, or by decreasing the memory requirements in his software model.

PERM can be used to rapidly assess the ability of a processor architecture of interest to support a particular algorithm or processing load. The architecture and software algorithm are rapidly cast into PERM structures, and then PERM is run against a variety of input data sizes until it "breaks." By looking at what breaks first and why -- that is, what resource is exhausted first, and as a result of which Segment or combination of Segments -- the analyst obtains valuable information about the robustness of the proposed system and its ability to achieve the desired processing goals.

PERM can also be used to challenge claims of run time performance. Such claims are often based on processor MIPS or FLOPS rates that are only achievable in short bursts on ideally suited problems. By taking the analysis one step down, PERM can force the analyst to examine such things as actual memory bandwidth, or synchronization delays in a distributed processing environment. In this way, PERM can help the analyst obtain a defensible estimate of the size of these parameters.

1.7 LIMITATIONS

PERM is not a high fidelity modeling tool, and as such the results that it produces must not be used in ways that are inappropriate. Further, PERM is only in a prototype stage. We are aware of improvements that can be made in the modeling strategy and in the user interface. This section will list the limitations we are aware of, and some of the areas where improvement could be made. Hopefully, these will be incorporated into subsequent PERM releases.

1.7.1 Transfer Function Coefficients

PERM relies completely on the judgement and experience of the analyst in defining the Transfer Functions that are the heart of the PERM model. It is easy to make up bad Transfer Functions; it is much harder, and more time consuming, to make up good ones. In our work on the project, we have already gained a good deal of experience in how to derive, and document, Transfer Functions, but it is still very much of an acquired art rather than a science. Anyone using PERM, or considering analyses based on it, must be aware that the analyst, in effect, has total control over the PERM output by way of the Transfer Functions. Or, to state the same matter another way, if one wishes to have a sense of how reliable the outputs of PERM are, he should look closely at the Transfer Functions and the techniques the analyst used to estimate or derive them.

1.7.2 Memory Utilization in Join Segments

The fact that Join Segments have no resource utilization Transfer Functions makes for a certain awkwardness in using the tool. For, whenever a processor enters a Join Segment, memory capacity utilization automatically drops to zero, even if the process has just gone into a halt state temporarily until synchronization. Some "fixes" have been proposed, but they rely on unintuitive gimmicks, and are not very satisfying. This is something that needs to be looked at in up-coming releases.

1.7.3 Fidelity

While PERM requires no assumptions about the units of time appropriate to its simulation, the user will sense that we are, in general, talking about "blocks of code" that might run on the order of 10's of milliseconds to a few seconds. If that is accurate, then Segments that run for a few microseconds will be lost in the noise; or alternatively, Segments whose model calls for Run Times on the order of 10s of seconds will dominate all others. The point is that the Run Times of the Segments that make up a Task Class should all be more or less compatible -- say, within three orders of magnitude of each other. This, in turn, may guide the analyst in what he chooses to model; that is, how many Segments he uses to model a Section of code, and the amount of time he spends researching and defending his Transfer Function estimates.

1.7.4 Task Data Size Parameters

As we have noted, Transfer Functions accept a Data Size value as input, and produce a resource utilization value as output, based on the coefficients entered by the analyst during Segment Class definition. These Data Sizes are then provided at the Task level; every Segment in the Task uses the same Data Size value.

This approach is subject to criticism in two ways. First, the particular form of the Transfer Function does not cover all cases. We can all think, for example, of N**3 algorithms, or of algorithms that depend on two data sizes. Second, it is a somewhat arbitrary condition to insist that all Segments in a Task Class must use the same Data Size. Even though the Reduction Factor, R, can be used to distribute the Data across the array (that is, for example, choosing R = .25 if there are four processors), there are nevertheless cases where it will not be immediately apparent what is the best way to model the situation within the PERM structures.

1.7.5 Pipelined Architectures

A similar thing happens when PERM is used to model a highly pipelined, queue/server type of system. Segments tend to be best at modeling larger blocks of code, rather than many smaller, event-driven blocks of code. In the AOA/AOSP modeling endeavor, several "tricks" were developed to model the desired behavior, and the general consensus was that PERM was remarkably successful. Nevertheless, such architectures are not PERM's natural domain, and a *good deal of analyst ingenuity may be required.*

1.7.6 PERM is Still a Prototype

Finally, and perhaps most important, it must be remembered that there is still very little experience in using PERM on real systems and real problems. Thus, it is still much to early to decide on its usefulness. There is good reason to think that it can fill an important niche between expensive low-level event driven simulations and back-of-the-envelope "guesstimation", but only a great deal of additional experience will decide that question.

2. USER'S GUIDE

The purpose of this section is to provide specific information on how to install and run PERM (Section 2.1), on the current status of the PERM software (Section 2.2), and on the PERM commands (Section 2.3).

2.1 PERM CONFIGURATION AND FILE CONSIDERATIONS

2.1.1 What is Necessary to Run PERM?

PERM is intended to run on IBM PC/AT computers, or compatibles, under the DOS 3.3 operating system. PERM has been run successfully with both 286 and 386 processors, running at both 12.5 and 20 MHz. It expects an EGA or VGA color monitor, or a Hercules monochrome monitor. It supports a variety of printers (see the Printer command in DPERM for the current list of printer drivers), and it expects the printer to be attached to the parallel output port LPT1.

A current limitation on PERM is that it is memory-bound. It does not currently support extended memory, and it requires a full 640K of core memory to operate efficiently. Even this is not really enough to be comfortable, and IPERM continuously displays a "Memory Remaining" message reminding the user of the amount of unused space still available.

[Recent detailed tests have shown that the reported "Space Remaining" is about 20KBytes too large. Correction of this anomaly must await future releases of PERM.]

To keep as much memory as possible available for PERM, it is recommended that PERM not be run with any TSR (Terminate and Stay Resident) programs. Again, this is not a hard requirement; PERM has been run successfully with TSRs, but they do use up memory, and that leaves less for the user to build PERM data structures.

2.1.2 PERM Files

PERM software consists of three executable DOS files:

* iperm.exe
* cperm.exe
* dperm.exe

In addition, there are three pairs of Help files to support the PERM on-line Help capability. They are:

* iperm.hlp,	iperm.ndx
* cperm.hlp,	cperm.ndx
* dperm.hlp,	dperm.ndx

These are specially pre-processed ASCII files. They must be in the default directory whenever the corresponding PERM executable is invoked.

In addition, several test case files are provided, matching the test case presented in Section 1 and in Appendix A. They are:

*	testcase.vpe	[a verified Processor Ensemble, suitable for use by the Load command]
*	test_trk.vtc	[three verified Task Classes, suitable for use test_eph.vtc by the Loadcommand]
	test_srt.vtc	
*	testcase.lod	[a verified System Load, suitable for use by the Load command]
*	testcase.inx testcase.pe testcase.ehf	[the output from CPERM, based on the System Load contained in testcase.lod, and suitable for use by DPERM]

When PERM is used, the analyst will be creating additional files: files to store PERM data structures for later use, text files to review the contents of the PERM data structures, log files to record PERM error messages or other text written to the screen, etc. In most cases, the user can specify the complete DOS file name, including the drive/directory path. However, in the hand-off between CPERM and DPERM, three intermediate files are created by CPERM in the default directory, and are looked for by DPERM in the default directory. The user can specify a file name, but not the extension, and not the drive or directory. This is explained at greater length in the CPERM Compute command and the DPERM Load command.

2.2 CURRENT STATUS OF PERM SOFTWARE

In this section, the current status of the PERM software will be described, including the implications for potential users. PERM was developed under a very aggressive schedule. Less than one man year, over a period of less than six months, was devoted to the development of the over 35,000 lines of executable code (80,000 total lines) that constitute the three PERM modules. This has several consequences for the PERM software.

First, it meant forgoing the luxury of a detailed design period. Coding began almost immediately, and thus PERM was developed in a design-as-you-go fashion.

Second, it has meant that a full testing program was not possible. PERM has been used by a number of different persons on the PERM development program to build and execute models, and all known bugs have been fixed. Thus, we feel safe in characterizing PERM as of good quality for internal use. However, it is not commercial quality, and it is inevitable that additional errors will crop up.

PERM was built with extensive self-diagnosis and checking code. Should PERM itself discover an apparent error, a System Error screen is displayed with debugging information. If such a screen is encountered, it is probably best to save work and exit the tool; it is not safe to continue to use the tool, since a possibly fatal error condition has been detected. Further, data saved following a system error should be treated as suspect.

Third, PERM was developed using two PC tool sets. These were purchased with Government funds, and delivered to the Government together with PERM software and documentation. They are required for recompilation (fixing PERM bugs), but are no longer with the developers. Thus, it is not immediately apparent how PERM bugs will be corrected when they are discovered.

Fourth and finally, PERM must be considered as being in a prototype phase. It can be used, but many potential improvements have already been identified, and plans for future versions of the tool are under discussion. It is not certain that data structures created using the current version of the tool will necessarily be binary compatible with future versions, should they be developed.

2.3 PERM EXECUTION AND COMMANDS

In this section, we provide detailed guidance on how to run PERM modules, and on the commands that drive the PERM system. For each of the three major modules -- IPERM, CPERM, and DPERM, we indicate how to execute the modules from the DOS environment (that is, what files are required, and where they should reside). Then, we provide a complete listing of the on-line Help files provided for every PERM command.

2.3.1 IPERM

2.3.1.1 Running IPERM – In order to execute IPERM, do the following things:

1. Place yourself in a directory. This will be your default directory. Make sure that the files iperm.hlp and iperm.ndx are in this directory. If the file iperm.exe is not in this directory, or if any of the other files (holding Processor Ensemble, Task Class, or System Load data structures) are not in this directory, be sure you know where they are, so you can specify them with their full DOS path name.

2. Enter iperm.exe, including its full DOS path if it is not in your current, default, directory.

2.3.1.2 Help Files of IPERM – This section lists the on-line Help files provided for the IPERM commands. On-line Help is always available using the <F1> key. These are broken into four groups:

- Processor Ensemble Commands
- Task Class Commands
- System Load Commands
- miscellaneous Commands (Log, Quit, etc.)

2.3.1.2.1 Help Files for Processor Ensemble Operations

Processor Ensemble Operations (@@P)

The command opens into a tree of sub-menus for building and editing a Processor Ensemble data structure. When you enter this command, one of two situations will exist:

(1) you already have a PE data structure in memory (as a result of a previous Create or Load command); or

(2) you don't have a PE data structure in memory.

In case (1), you will immediately be able to use the commands available at the next lower menu. In case (2), you must first either Create or Load a PE data structure before any of the other commands will function.

Create Processor Ensemble (@@PC)

If there is currently no Processor Ensemble data structure in memory (either because you just entered PERM, or because you used the Remove command) (see @@PR), there are two ways of getting one into memory. You can use the Load command (see @@PL) (which retrieves a PE data structure that has been previously stored to disk using the Save) (command (@@PS); or, you can use the Create command.

You cannot use the Create command if a PE data structure already exists in memory. To get rid of that data structure, you must use the Remove command.

NOTE:

> The Remove command does not ask the user to "Confirm" the decision, and immediately and irrevocably erases the data. Thus, if there is any chance that the data currently in memory may be needed, it is best to first save it to permanent storage (disk) using the Save command.

When you exercise the Create command, you will be asked for a name (up to 32 characters, including spaces and special characters if you wish) for the PE data structure, and for your name (author). Other information concerning verify status and dates is displayed, but is not editable. Once you have created a PE data structure using the Create command, you can then edit it or use any of the other PE commands on it.

NOTE:

PERM will only accept (operate on) one PE data structure at a time.

Edit Processor Ensemble (@@PE)

Use of this command opens onto a series of sub-menus that permit the major structures of a Processor Ensemble to be created and manipulated. The command, and its sub-commands, operate on the PE data structure that is currently in memory (as the result of a Create or Load command).

Remove Processor Ensemble (@@PR)

This command immediately clears PERM memory so that the Create or Load commands can be exercised.

NOTE:

The Remove command does not ask the user to "Confirm" the decision, and immediately and irrevocably erases the data. Thus, if there is any chance that the data currently in memory may be needed, it is best to first save it to permanent storage (disk) using the Save command.

Display Processor Ensemble (@@PD)

Use of this command causes a scrolling ASCII file to be sent to the user's terminal. The user scrolls through the file one page at a time using (for example) the <space> bar to move to the next page (as in the DOS "more" command). When the last page appears, so does the PE menu on top of it. To remove the menu so that the data can be seen, use <ESC> to move up to the top menu. The PE menu can then be re-entered without losing any data.

The organization of the data constituting the PE data structure is as follows:

First, the Processor Ensemble name and validation status
Second, Processor Classes (one for each Class)
    Processor Class name

- accessible processors variable list
- accessible memories variable list
- accessible busses variable list

Instantiations (one for each instantiation)

Instantiation name

- names for accessible processors
- names for accessible memories
- names for accessible busses

Third, Memory Classes (one for each Class)

Memory Class name

Memory Size (Bytes)

Memory Bandwidth (Bytes/Second)

- accessible processors variable list

Instantiations

Instantiation name

- names for accessible processors

Fourth, Bus Classes (one for each Class)

Bus Class name

Bus Bandwidth (Bytes/Second)

- accessible processors variable list

Instantiations

Instantiation name

- names for accessible processors

As each page is displayed, the user can elect to ignore the remaining data by selecting <c>; use of any other key will scroll the next page.

There is another way to view the PE data structure. The user can print the same data to a DOS file using the Print command, and can then exit PERM and use standard DOS utilities (print, editor etc.) to review the data. If this is done, however, the data should first be saved using the Save command, since the PE data structure will be lost once PERM is exited. Thus, an advantage of the Display command is that it enables the user to see the current status of the PE structure without having to exit PERM.

Load a Processor Ensemble (@@PL)

The PERM commands operate on a Processor Ensemble data structure in memory. There can only be one such data structure in memory at a time. There are two ways of getting a PE data structure into memory: first, by using the Create command; and second, by using the Load command.

The Load command will Load a PE data structure into memory from a DOS file -- one that has been previously created using the Save command. Thus, for example, a user can first Create a PE data structure and work on it for a while; then save that work to a DOS file using the Save command. At any later time (even after having first exited PERM and then restarted), the saved work can be brought back into memory to be further worked on using the Load command.

The user will be prompted for the name of a DOS file. A full path name (including drive/directory) can be included. If that is omitted, the current default directory (the directory from which PERM was executed) will be used. If the file cannot be found or opened, PERM informs the user, and waits for further instructions. If you try to Load a file that was not created using the Save command (that is, a file that does not conform to PERM's internal data structures), PERM will detect that fact, and won't load the file.

You cannot use the Load command if a PE data structure already exists in memory. The way to get rid of such a data structure is to first remove it using the Remove command.

NOTE:

> The Remove command does not ask the user to "Confirm" the decision, and immediately and irrevocably erases the data. Thus, if there is any chance that the data currently in memory may be needed, it is best to first save it to permanent storage (disk) using the Save command.

Also, just for completeness, we note that there is a big difference between a file created using the Save command and a file created using the Print command. The file created by Save is in a form that can be re-loaded and operated on via the Load command. A file created using the Print command is simply an ASCII text file that can be printed or edited using DOS utilities and programs. Thus, it would be wise to keep the two distinct by using the DOS file modifier -- say "xxx.DAT" for loadable files, and "xxx.TXT" for ASCII files. If will also prevent you from inadvertently overwriting a loadable file with an ASCII file, or visa-versa.

Save Processor Ensemble (@@PS)

The purpose of this command is to enable the user to preserve work in a permanent form so that it can easily be recalled for later use. The command writes the PE data structure to a DOS file; the data can subsequently be recalled by using the Load command.

It is not necessary that the full PE data structure be complete. Whatever has been created will be saved -- even quite partial and incomplete versions. Thus, the user can execute the Save command frequently during an editing session. If the system should fail, the data will have been stored to disk, and can be recalled for further use at the point at which it was saved.

The user will be prompted for a DOS file name. The full pathname can be included (drive/directory); if it is omitted, the current default directory will be used for the file (the directory from which PERM was executed). If PERM cannot open the file, it will notify the user, and wait for further instructions. If the file already exists, it will be overwritten (and hence, lost).

In the Task Class operations menu, there is also a Save command. If you should inadvertently use one of these two Save commands with a file name that already exists, you will lose that file (that is, the file will be overwritten). Thus, it is probably a good idea to have a file naming convention that will keep you from getting mixed-up about which are your ASCII files, your PE files, and your Task Class files.

Also, just for completeness, we note that there is a big difference between a file created using the Save command and a file created using the Print command. The file created by Save is in a form that can be re-loaded and operated on via the Load command. A file created using the Print command is simply an ASCII text file that can be printed or edited using DOS utilities and programs. A thoughtfully chosen naming convention can help to prevent inadvertent errors and loss of data.

Print Processor Ensemble (@@PP)

The purpose of this command is to create an ASCII file containing the complete state of the Processor Ensemble data structure currently in memory. The file is identical to the file that the Display command routes to the terminal. The order of presentation for the data is:

First, the Processor Ensemble name and validation status
Second, Processor Classes (one for each Class)
    Processor Class name
        - accessible processors variable list
        - accessible memories variable list
        - accessible busses variable list

Instantiations (one for each instantiation)
Instantiation name
- names for accessible processors
- names for accessible memories
- names for accessible busses

Third, Memory Classes (one for each Class)
Memory Class name
Memory Size (Bytes)
Memory Bandwidth (Bytes/Second)
- accessible processors variable list

Instantiations
Instantiation name
- names for accessible processors

The user will be prompted for the name of a DOS file. A full path name (including arive/directory) can be included. If that is omitted, the current default directory (the directory from which PERM was executed) will be used. If the file cannot be found or opened, PERM informs the user, and waits for further instructions.

Also, just for completeness, we note that there is a big difference between a file created using the Save command and a file created using the Print command. The file created by Save is in a form that can be re-loaded and operated on via the Load command. A file created using the Print command is simply an ASCII text file that can be printed or edited using DOS utilities and programs. Thus, it would be wise to keep the two distinct by using the DOS file modifier -- say "xxx.DAT" for loadable files, and "xxx.TXT" for ASCII files. If will also prevent you from inadvertently overwriting a loadable file with an ASCII file, or visa-versa.

After execution of the Print command, the file that has been created can be operated on by DOS utilities -- the print command, an editor, etc. To do this, the user must exit PERM. Remember -- if you will want the data for later use, you must first save it using the Save command.

Verify the Processor Ensemble (@@PV)

The purpose of this command is to ensure the internal consistency of the PE data structure as it has been entered by the user. In particular, consistent use of Class and Instantiation names is validated, both verifying the correct spelling of multiple uses of the name (the names entered with the Create and Instantiate commands being given preference); and validating that all class names, and all instantiations within a class, are unique. Any validation errors are reported in a summary message scrolled onto the terminal; thus, the user is given specific information about the source of the error so that it can be corrected.

NOTE:

> The user may wish to use the PERM Log facilities (top-level commands) if the error list is too long to remember easily. If a Log file is open, PERM will write the error message not only to the screen, but also to the Log file. That file can then be accessed (via DOS programs and utilities), for example, to obtain a hard-copy.
>
> A validated PE data structure is required before the user can proceed to the second stage of system definition -- Task Class operations. This is because the Task Class operations require access to information about the hardware on which the Segments, Threads, and Tasks that constitute the PERM software structures will run. If a PE data structure has already exists (in a DOS file) and has been loaded into memory (using the Load command), it can then be verified (using the Verify command), and the user can then proceed to Task Class definition.

Edit Processor Ensemble Header (@@PEH)

This command will enable the user to view and/or change the Processor Ensemble name and the name of the author. Other information, on verify status, is also displayed, but is not editable. One might use this command, for example, to begin with an older version of a PE data structure and then alter it (i.e., edit it) for a new processor configuration.

Edit Processors (@@PEP)

This command opens onto a series of sub-menus that permit the user to create, modify and display Processor Classes and their Instantiations.

NOTE:

> Use of PERM requires familiarity with the underlying PERM concepts and modeling strategy. The sub-menus that open out from this command are essentially for data entry. The user must already have identified the major Processor Ensemble data structures, and selected appropriate naming conventions. Thus, PERM assumes that a fair amount of preparatory work has been done prior to this, the data entry stage of using PERM.

Generally speaking, the correct order for using the PE data structure commands is to first create all classes (and validate their accuracy), and then instantiate them. The reason is that Classes cannot be edited (except deleted) once they have been instantiated. A corollary of this rule is that all Classes -- Processor, Memory, and Bus -- be created and completed before any are instantiated. While PERM does not enforce this approach, it is the recommended one.

Edit Memories (@@PEM)

This command opens onto a series of sub-menus that permit the user to create, modify and display Memory Classes and their Instantiations.

NOTE:

Use of PERM requires familiarity with the underlying PERM concepts and modeling strategy. The sub-menus that open out from this command are essentially for data entry. The user must already have identified the major Processor Ensemble data structures, and selected appropriate naming conventions. Thus, PERM assumes that a fair amount of preparatory work has been done prior to this, the data entry stage of using PERM.

Generally speaking, the correct order for using the PE data structure commands is to first create all classes (and validate their accuracy), and then instantiate them. The reason is that Classes cannot be edited (except deleted) once they have been instantiated. A corollary of this rule is that all Classes -- Processor, Memory, and Bus -- be created and completed before any are instantiated. While PERM does not enforce this approach, it is the recommended one.

Edit Busses (@@PEB)

This command opens onto a series of sub-menus that permit the user to create, modify and display Bus Classes and their Instantiations.

Create a Processor Class (@@PEPA)

This command creates the top header (name) for a Processor Class. Names entered will appear in lower-level scrolling displays to be selected by the user for editing, etc. That is, the first step in working with a Processor Class is to create it; only then can it be edited.

Like all other names in PERM, the Processor Class name can be up to 32 characters, and can contain blanks and other special characters if desired.

NOTE:

PERM will not keep you, at this stage, from re-using the same name for two different classes. However, this should not be done, and will be detected as an error when the Verify function is executed (top menu). Also, the effects on editing classes with the same name will be unpredictable. Thus, use distinct names for all Processor Classes.

Edit a Processor Class (@@PEPB)

Once a processor class has been created, it can be operated on by the other commands -- Edit, Delete, (See @@PEPC) and Display (See @@PEPD). This command then opens onto a sub-menu which enables the user to create connectivities to other Processor, Memory, and Bus Classes.

Delete a Processor Class (@@PEPC)

This command immediately and irrevocably deletes a selected Processor Class from the current Processor Ensemble data structure.

NOTE:

This command will execute even if there exist instantiations of this Processor Class. They will be lost, along with all connectivity variable definitions. Also, no "confirm" is required; once the Class is selected and <Return> is pressed, the selected Class is immediately deleted. Thus, use this command with care.

Display all Processor Classes (@@PEPD)

This command operates in a similar manner to the Display option at the top-most menu. All Processor Class information, including any Instantiations, is sent in ASCII for to the terminal for inspection. The user scrolls through the file one page at a time using (for example) the <space> bar to move to the next page (as in the DOS "more" command). When the last page appears, so does the menu on top of it. To remove the menu so that the data can be seen, use <ESC> <ESC> <ESC> to move up to the top menu.

The organization of the data constituting the Processor Class display is as follows:

- First, the Processor Ensemble name and validation status
- Second, Processor Classes (one for each Class)
    - Processor Class name
        - accessible processors variable list
        - accessible memories variable list
        - accessible busses variable list
    - Instantiations (one for each instantiation)
        - Instantiation name
            - names for accessible processors
            - names for accessible memories
            - names for accessible busses

As each page is displayed, the user can elect to ignore the remaining data by selecting <c>; use of any other key will scroll the next page.

Instantiate a Processor Class (@@PEPE)

As opposed to Processor Classes, which generically describe the connectivity of an abstract processor to abstract processors, memories, and busses, an Instantiation creates a "real" instance of the class -- an object that inherits the connectivity properties of its class. This command simply allows the user to give a name to the instantiation; its specific properties (that is, its connectivity to other instantiated entities) is then subsequently established using the "Edit Instantiation" command.

When executed, the command will first prompt the user to select which of the existing Processor Classes is to be instantiated. The current Processor Classes will be presented in a list which is scrolled using the <Home> (up) and <End> (down) keys. When the correct entry is high-lighted, press <Return> to select it. You will then be asked for the name of the instantiation. Up to 32 characters are permitted, including blanks and special characters. The command can be exited at any time without change to the PE data structure by pressing <ESC>.

NOTE:

PERM will not keep you, at this stage, from re-using the same name for two different instantiations. However, this should not be done, and will be detected as an error when the Verify function is executed (top menu). Also, the effects on editing instantiations with the same name will be unpredictable. Thus, use distinct names for all Processor Class Instantiations.

> Generally speaking, the correct order for using the PE data structure commands is to first create all classes (and validate their accuracy), and then instantiate them. The reason is that Classes cannot be edited (except deleted) once they have been instantiated. A corollary of this rule is that all Classes -- Processor, Memory, and Bus -- be created and completed before any are instantiated. While PERM does not enforce this approach, it is the recommended one.

Edit a Processor Class Instantiation (@@PEPF)

Once a Processor Class has been instantiated (using the "Instantiate" command), its specific connectivity properties can be specified.

Every processor instantiation belongs to a Processor Class (that was selected when the instantiation was entered). That class, in turn, specifies by means of variable names the connectivity of that class -- that is, the other Processor, Memory, and Bus Classes a processor (of that Class) is connected to. Thus, for example, a series of Associated Processor variables was created along with the Processor Class, each of those variables, in turn, being associated with a Processor Class of their own. In an instantiation, the variable names are replaced by the names of instantiations of Processors belongs to the variable's class.

Thus, to edit an instantiation, the user must select the following things:

(1) the type of PE entity (Processor, Memory, or Bus) whose connectivity variable is to be specified;

(2) what Class of that type the variable belongs to;

(3) what is the name of the variable belonging to that Class; and

(4) the name of the instantiated entity to be assigned to the variable.

The information corresponding to (1) will be selected in the sub-menu that opens under this command. The information for (2) and (3) is then selected by scrolling menus using the <Home> and <End> keys; and the information in (4) is entered by keying in the desired name.

Delete a Processor Class Instantiation (@@PEPG)

This command permits existing instantiations to be deleted from the Processor Ensemble data structure. The user will first be prompted for the Processor Class the instantiation belongs to, which is selected from the scrolling window using the <Home> and <End> keys (and <Return> to select). Then, the existing instantiations from this Class will be presented, and the one to be deleted is selected in the same manner. If the user decides not to delete anything, the <ESC> key will exit the command without altering the PE data structure.

Display All Instantiations of a Processor Class (@@PEPH)

This command works similarly to the Display commands at higher menus. However, only the information related to a single Processor Class and its instantiations is displayed. The user selects that class from a scrolling window using the <Home> and <End> keys (and <Return> to select). All information relating to that Processor Class, including all Instantiations, is sent in ASCII form to the terminal for inspection. The user scrolls through the file one page at a time using (for example) the <space> bar to move to the next page (as in the DOS "more" command). When the last page appears, so does the menu on top of it. To remove the menu so that the data can be seen, use <ESC> <ESC> <ESC> to move up to the top menu.

The organization of the data constituting the Processor Class Instantiation display is as follows:

Processor Class name
- accessible processors variable list
- accessible memories variable list
- accessible busses variable list

Instantiations (one for each instantiation)
  Instantiation name
    - names for accessible processors
    - names for accessible memories
    - names for accessible busses

As each page is displayed, the user can elect to ignore the remaining data by selecting <c>; use of any other key will scroll the next page.

Create a Memory Class (@@PEMA)

This command creates the top header (name) for a Memory Class. Names entered will appear in lower-level scrolling displays to be selected by the user for editing, etc. That is, the first step in working with a Memory Class is to create it; only then can it be edited.

Like all other names in PERM, the Memory Class name can be up to 32 characters, and can contain blanks and other special characters if desired.

NOTE:

> PERM will not keep you, at this stage, from re-using the same name for two different classes. However, this should not be done, and will be detected as an error when the Verify function is executed (top menu). Also, the effects on editing classes with the same name will be unpredictable. Thus, use distinct names for all Memory Classes.

In addition to the name of the Memory Class, the user is prompted for two performance characteristics: the total size (capacity) of the memory (in Bytes); and the memory bandwidth (in Bytes/Second). These are entered as integers, without commas; entering a value in the wrong format will result in a error message, and a prompt to re-enter the value. Both memory size and bandwidth have default values of 1. All instantiations of a Memory Class inherit the size and bandwidth characteristics of the class to which they belong.

The memory bandwidth ordinarily will not include addresses -- only the number of bytes that can be transferred by Read and Write operations on the memory. Transfer functions in code Segments associated with processors that access the memory will specify the amount of bandwidth the segment requires; these are then summed across all processors that access the memory, and the total bandwidth requirement can be compared (at any point in time) to the total available bandwidth. Similarly, memory space requirements (summed across all active segments) can be compared to the available memory.

Edit a Memory Class (@@PEMB)

Once a memory class has been created, it can be operated on by the other commands -- Edit, Delete, and Display. This command then opens onto a sub-menu which enables the user to create connectivities to Processor Classes.

Delete a Memory Class (@@PEMC)

This command immediately and irrevocably deletes a selected Memory Class from the current Processor Ensemble data structure.

NOTE:

> This command will execute even if there exist instantiations of this Memory Class. They will be lost, along with all connectivity variable definitions. Also, no "confirm" is required; once the Class is selected and <Return> is pressed, the selected Class is immediately deleted. Thus, use this command with care.

Display all Memory Classes (@@PEMD)

This command operates in a similar manner to the Display option at the top-most menu. All Memory Class information, including any Instantiations, is sent in ASCII form to the terminal for inspection. The user scrolls through the file one page at a time using (for example) the <space> bar to move to the next page (as in the DOS "more" command). When the last page appears, so does the menu on top of it. To remove the menu so that the data can be seen, use <ESC> <ESC> <ESC> to move up to the top menu.

The organization of the data constituting the Memory Class display is as follows:

First, the Processor Ensemble name and validation status
Second, Memory Classes (one for each Class)
    Memory Class name
    Memory Size (Bytes)
    Memory Bandwidth (Bytes/Second)
        - accessible processors variable list
        Instantiations
            Instantiation name
            - names for accessible processors

As each page is displayed, the user can elect to ignore the remaining data by selecting <c>; use of any other key will scroll the next page.

Instantiate a Memory Class (@@PEME)

As opposed to Memory Classes, which generically describe the connectivity of an abstract memory to abstract processors, an Instantiation creates a "real" instance of the class -- an object that inherits the connectivity properties of its class. This command simply allows the user to give a name to the instantiation; its specific properties (that is, its connectivity to other instantiated entities) is then subsequently established using the "Edit Instantiation" command.

When executed, the command will first prompt the user to select which of the existing Memory Classes is to be instantiated. The current Memory Classes will be presented in a list which is scrolled using the <Home> (up) and <End> (down) keys. When the correct entry is high-lighted, press <Return> to select it. You will then be asked for the name of the instantiation. Up to 32 characters are permitted, including blanks and special characters. The command can be exited at any time without change to the PE data structure by pressing <ESC>.

NOTE:

PERM will not keep you, at this stage, from re-using the same name for two different instantiations. However, this should not be done, and will be detected as an error when the Verify function is executed (top menu). Also, the effects on editing instantiations with the same name will be unpredictable. Thus, use distinct names for all Memory Class Instantiations.

Generally speaking, the correct order for using the PE data structure commands is to first create all classes (and validate their accuracy), and then instantiate them. The reason is that Classes cannot be edited (except deleted) once they have been instantiated. A corollary of this rule is that all Classes -- Processor, Memory, and Bus -- be created and completed before any are instantiated. While PERM does not enforce this approach, it is the recommended one.

Edit a Memory Class Instantiation (@@PEMF)

Once a Memory Class has been instantiated (using the "Instantiate" command), its specific connectivity properties can be specified.

Every memory instantiation belongs to a Memory Class that was selected when the instantiation was entered. That class, in turn, specifies by means of variable names the connectivity of that class -- that is, the other Processor Classes a memory is connected to. Thus, for example, a series of Associated Processor variables was created along with the Memory Class, each of those variables, in turn, being associated with their own Processor Class. In an instantiation, the variable names are replaced by the names of instantiations of processors belonging to the variable's class.

Thus, to edit an instantiation, the user must select the following things:

(1) what Processor Class the variable belongs to;
(2) what is the name of the variable belonging to that Class; and
(3) the name of the instantiated processor to be assigned to the variable.

This information will be entered from the sub-menu that follows this command, which also permits the user to edit the name of the instantiation (Header). The information for (1) and (2) is then selected by scrolling menus using the <Home> and <End> keys; and the information in (3) is entered by keying in the desired name.

Delete a Memory Class Instantiation (@@PEMG)

This command permits existing instantiations to be deleted from the Processor Ensemble data structure. The user will first be prompted for the Memory Class the instantiation belongs to, which is selected from the scrolling window using the <Home> and <End> keys (and <Return> to select). Then, the existing instantiations from this Class will be presented, and the one to be deleted is selected in the same manner. If the user decides not to delete anything, the <ESC> key will exit the command without altering the PE data structure.

Display All Instantiations of a Memory Class (@@PEMH)

This command works similarly to the Display commands at higher menus. However, only the information related to a single Memory Class and its instantiations is displayed. The user selects that class from a scrolling window using the <Home> and <End> keys (and <Return> to select). All information relating to that Memory Class, including all Instantiations, is sent in ASCII form to the terminal for inspection. The user scrolls through the file one page at a time using (for example) the <space> bar to move to the next page (as in the DOS "more" command). When the last page appears, so does the menu on top of it. To remove the menu so that the data can be seen, use <ESC> <ESC> <ESC> to move up to the top menu.

The organization of the data constituting the Memory Class Instantiation display is as follows:

Memory Class name
Memory Size (Bytes)
Memory Bandwidth (Bytes/Second)
- accessible processors variable list
Instantiations
Instantiation name
- names for accessible processors

As each page is displayed, the user can elect to ignore the remaining data by selecting <c>; use of any other key will scroll the next page.

Create a Bus Class (@@PEBA)

This command creates the top header (name) for a Bus Class. Names entered will appear in lower-level scrolling displays to be selected by the user for editing, etc. That is, the first step in working with a Bus Class is to create it; only then can it be edited.

Like all other names in PERM, the Bus Class name can be up to 32 characters, and can contain blanks and other special characters if desired.

NOTE:

PERM will not keep you, at this stage, from re-using the same name for two different classes. However, this should not be done, and will be detected as an error when the Verify function is executed (top menu). Also, the effects on editing classes with the same name will be unpredictable. Thus, use distinct names for all Bus Classes.

In addition to a name for the Bus Class, the user will be prompted for a total Bus Bandwidth (in Bytes/Second). This is entered as an integer, without commas; entering a value in the wrong format will result in an error message, and a prompt to re-enter the value. Bus bandwidth has a default value of 1.

All instantiations of a Bus Class inherit the bandwidth characteristics of that class. Each code segment on a processor that can access the bus will contain a transfer function specifying the amount of bandwidth required by the segment. These are summed across active segments (at any point in time) to obtain the total bandwidth requirement, and this requirement is then compared against the actual bandwidth capacity of the bus.

Edit a Bus Class (@@PEBB)

Once a bus class has been created, it can be operated on by the other commands -- Edit, Delete, and Display. This command then opens onto a sub-menu which enables the user to create connectivities to Processor Classes.

Delete a Bus Class (@@PEBC)

This command immediately and irrevocably deletes a selected Bus Class from the current Processor Ensemble data structure.

NOTE:

> This command will execute even if there exist instantiations of this Bus Class. They will be lost, along with all connectivity variable definitions. Also, no "confirm" is required; once the Class is selected and <Return> is pressed, the selected Class is immediately deleted. Thus, use this command with care.

Display all Bus Classes (@@PEBD)

This command operates in a similar manner to the Display option at the top-most menu. All Bus Class information, including any Instantiations, is sent in ASCII form to the terminal for inspection. The user scrolls through the file one page at a time using (for example) the <space> bar to move to the next page (as in the DOS "more" command). When the last page appears, so does the menu on top of it. To remove the menu so that the data can be seen, use <ESC> <ESC> <ESC> to move up to the top menu.

The organization of the data constituting the Bus Class display is as follows:

First, the Processor Ensemble name and validation status
Fourth, Bus Classes (one for each Class)

Bus Class name
Bus Bandwidth (Bytes/Second)
- accessible processors variable list
Instantiations
Instantiation name
- names for accessible processors

As each page is displayed, the user can elect to ignore the remaining data by selecting <c>; use of any other key will scroll the next page.

Instantiate a Bus Class (@@PEBE)

As opposed to Bus Classes, which generically describe the connectivity of an abstract bus to abstract processors, an Instantiation creates a "real" instance of the class -- an object that inherits the connectivity properties of its class. This command simply allows the user to give a name to the instantiation; its specific properties (that is, its connectivity to other instantiated entities) is then subsequently established using the "Edit Instantiation" command.

When executed, the command will first prompt the user to select which of the existing Bus Classes is to be instantiated. The current Bus Classes will be presented in a list which is scrolled using the <Home> (up) and <End> (down) keys. When the correct entry is high-lighted, press <Return> to select it. You will then be asked for the name of the instantiation. Up to 32 characters are permitted, including blanks and special characters. The command can be exited at any time without change to the PE data structure by pressing <ESC>.

NOTE:

PERM will not keep you, at this stage, from re-using the same name for two different instantiations. However, this should not be done, and will be detected as an error when the Verify function is executed (top menu). Also, the effects on editing instantiations with the same name will be unpredictable. Thus, use distinct names for all Bus Class Instantiations.

Edit a Bus Class Instantiation (@@PEBF)

Once a Bus Class has been instantiated (using the "Instantiate" command), its specific connectivity properties can be specified.

Every bus instantiation belongs to a Bus Class that was selected when the instantiation was entered. That class, in turn, specifies by means of variable names the connectivity of that class -- that is, the other Processor Classes a bus is connected to. Thus, for example, a series of Associated Processor variables was created along with the Bus Class, each of those variables, in turn, being associated with their own Processor Class. In an instantiation, the variable names are replaced by the names of instantiations of processors belonging to the variable's class.

Thus, to edit an instantiation, the user must select the following things:

(1) what Processor Class the variable belongs to;

(2) what is the name of the variable belonging to that Class; and

(3) the name of the instantiated processor to be assigned to the variable.

This information will be entered from the sub-menu that follows this command, which also permits the user to edit the name of the instantiation (Header). The information for (1) and (2) is then selected by scrolling menus using the <Home> and <End> keys; and the information in (3) is entered by keying in the desired name.

Delete a Bus Class Instantiation (@@PEBG)

This command permits existing instantiations to be deleted from the Processor Ensemble data structure. The user will first be prompted for the Bus Class the instantiation belongs to, which is selected from the scrolling window using the <Home> and <End> keys (and <Return> to select). Then, the existing instantiations from this Class will be presented, and the one to be deleted is selected in the same manner. If the user decides not to delete anything, the <ESC> key will exit the command without altering the PE data structure.

Display All Instantiations of a Bus Class (@@PEMH)

This command works similarly to the Display commands at higher menus. However, only the information related to a single Bus Class and its instantiations is displayed. The user selects that class from a scrolling window using the <Home> and <End> keys (and <Return> to select). All information relating to that Bus Class, including all Instantiations, is sent in ASCII form to the terminal for inspection. The user scrolls through the file one page at a time using (for example) the <space> bar to move to the next page (as in the DOS "more" command). When the last page appears, so does the menu on top of it. To remove the menu so that the data can be seen, use <ESC> <ESC> <ESC> to move up to the top menu.

The organization of the data constituting the Bus Class Instantiation display is as follows:

Bus Class name
Bus Bandwidth (Bytes/Second)
- accessible processors variable list
Instantiations
Instantiation name
- names for accessible processors

As each page is displayed, the user can elect to ignore the remaining data by selecting <c>; use of any other key will scroll the next page.

Edit the Header of a Processor Class (@@PEPBA)

Use of this command will allow the user to change the name of an existing Processor Class. The user scrolls through the existing Classes using the <Home> and <End> keys, and selects the desired one by entering <Return>. The new name can then be entered.

This command should be used with caution, since connectivities associated with Classes (Processor, Memory, and Bus) are entered by specifying Class names. If the Class name is changed, references to it in connectivity specifications must also be changed, or the PE structure will fail verification.

Add an Accessible Processor Variable (@@PEPBB)

The defining property of a Processor Class is its connectivity pattern -- that is, the list of other Processors, Memories, and Busses to which it is accessible. These connectivities are specified by means of variable names which act as place-holders until the Class is instantiated. When instantiated, the variable names are replaced by "real" names -- that is, names of other instantiated Processors, Memories, and Busses.

In PERM, for one Processor to be "accessible" to another means that threads running on those two processors can have inter-thread execution dependencies; heuristically, it means that the processors have some means of synchronizing -- for example, by means of a shared memory, a shared bus, or some low-level signalling capability.

As noted above, connectivities are specified using variable names as place holders. That is, a list of variable names is created. When a processor of the Class is instantiated, the variable names will be replaced with the instantiated names of "real" processors. The purpose of this command is to create the list of variable names, and to associate with each variable name a Processor Class.

The Verify command will validate that the Processor Class name associated with each variable is, in fact, the name of a Processor Class as entered via the PERM Processor Class Create or Edit Header commands. This is primarily a question of correct spelling. Also, it will be verified that the variable names are all unique within a given Class; duplicate variable names are not permitted, although PERM does not detect them at this point in the data entry process. Thus, use distinct variable names in the Accessible Processors list for each Processor Class.

When the command is executed, the user is presented with a menu of Processor Classes; the <Home> and <End> keys are used to scroll, and <Return> to select. Then, a menu is offered, providing one field for the variable name, and one for the Processor Class the variable is to belong to. The user can move freely between the fields using the <arrow> keys. The variable name can be up to 32 characters, including special characters. The name entered for the Processor Class should agree exactly (that is, character by character) with a Processor Class name as entered using the Create Processor Class command or the Edit Processor Class Header command. Failure to do so will be detected by the Verify command.

The command can be exited at any time using the <ESC> key without altering the PE data structure. Once <Return> is entered, the permanent PE structure is altered to contain whatever edits the user has entered.

Edit an Accessible Processor Variable (@@PEPBC)

The defining property of a Processor Class is its connectivity pattern -- that is, the list of other Processors, Memories, and Busses to which it is accessible. These connectivities are specified by means of variable names which act as place-holders until the Class is instantiated. When instantiated, the variable names are replaced by "real" names -- that is, names of other instantiated Processors, Memories, and Busses.

In PERM, for one Processor to be "accessible" to another means that threads running on those two processors can have inter-thread execution dependencies; heuristically, it means that the processors have some means of synchronizing -- for example, by means of a shared memory, a shared bus, or some low-level signaling capability.

As noted above, connectivities are specified using variable names as place holders. That is, a list of variable names is created. When a processor of the Class is instantiated, the variable names will be replaced with the instantiated names of "real" processors. The purpose of this command is to edit the list of variable names that has already been created (by use of the "Add Accessible Processor" command).

The Verify command will validate that the Processor Class name associated with each variable is, in fact, the name of a Processor Class as entered via the PERM Processor Class Create or Edit Header commands. This is primarily a question of correct spelling. Also, it will be verified that the variable names are all unique within a given Class; duplicate variable names are not permitted, although PERM does not detect them at this point in the data entry process. Thus, use distinct variable names in the Accessible Processors list for each Processor Class.

When the command is executed, the user is first presented with a menu of Processor Classes; the <Home> and <End> keys are used to scroll, and <Return> to select. Then, the current list of variables associated with that Processor Class is presented; scrolling and selection operate in the same way. Once the class and variable names have been selected, a menu is presented with the current data -- both the variable name, and the Processor Class associated with the variable.

NOTE:

One important use for this command is to correct spelling errors in the Processor Class field.

The user can move freely between the fields using the <arrow> keys. The variable name can be up to 32 characters, including special characters. The name entered for the Processor Class should agree exactly (that is, character by character) with a Processor Class name as entered using the Create Processor Class command or the Edit Processor Class Header command. Failure to do so will be detected by the Verify command.

The command can be exited at any time using the <ESC> key without altering the PE data structure. Once <Return> is entered, the permanent PE structure is altered to contain whatever edits the user has entered.

Delete an Accessible Processor Variable (@@PEPBD)

The purpose of this command is to delete a name from the list of variables designating the accessible processors associated with a Processor Class. These variables were previously created using the "Add Accessible Processor" command (or, edited using the "Edit Accessible Processor" command).

When the command is executed, the user is first presented with a menu of Processor Classes; the <Home> and <End> keys are used to scroll, and <Return> to select. Then, the current list of variables associated with that Processor Class is presented; scrolling and selection operate in the same way. Selection of the variable immediately results in its deletion from the current list of variables. The command can be exited without change to the PE data structure by entering <ESC>.

Add an Accessible Memory Variable (@@PEPBE)

The defining property of a Processor Class is its connectivity pattern -- that is, the list of other Processors, Memories, and Busses to which it is accessible. These connectivities are specified by means of variable names which act as place-holders until the Class is instantiated.

When instantiated, the variable names are replaced by "real" names -- that is, names of other instantiated Processors, Memories, and Busses.

In PERM, for a Memory to be "accessible" to a Processor means that, when a segment runs on that processor, it can make demands (both space and bandwidth) on that memory. These resource demands are specified by means of transfer functions -- two for each memory accessible to the processor. Likewise, when a Memory Class is defined, associated with it is a list of associated Processors. Thus, the association is specified twice: once processor-to-memory, and once memory-to-processor. The consistency of this dual specification is validated by the Verify command.

As noted above, connectivities are specified using variable names as place holders. That is, a list of variable names is created. When a processor of the Class is instantiated, the variable names will be replaced with the instantiated names of "real" memories. The purpose of this command is to create the list of variable names, and to associate with each variable name a Memory Class.

The Verify command will validate that the Memory Class name associated with each variable is, in fact, the name of a Memory Class as entered via the PERM Memory Class Create or Edit Header commands. This is primarily a question of correct spelling. Also, it will be verified that the variable names are all unique within a given Class; duplicate variable names are not permitted, although PERM does not detect them at this point in the data entry process. Thus, use distinct variable names in the Accessible Memories list for each Processor Class.

When the command is executed, the user is presented with a menu of Processor Classes; the <Home> and <End> keys are used to scroll, and <Return> to select. Then, a menu is offered, providing one field for the variable name, and one for the Memory Class the variable is to belong to. The user can move freely between the fields using the <arrow> keys. The variable name can be up to 32 characters, including special characters. The name entered for the Memory Class should agree exactly (that is, character by character) with a Memory Class name as entered using the Create Memory Class command or the Edit Memory Class Header command. Failure to do so will only be detected by the Verify command.

The command can be exited at any time using the <ESC> key without altering the PE data structure. Once <Return> is entered, the permanent PE structure is altered to contain whatever edits the user has entered.

Edit an Accessible Memory Variable (@@PEPBF)

The defining property of a Processor Class is its connectivity pattern -- that is, the list of other Processors, Memories, and Busses to which it is accessible. These connectivities are specified by means of variable names which act as place-holders until the Class is instantiated. When instantiated, the variable names are replaced by "real" names -- that is, names of other instantiated Processors, Memories, and Busses.

In PERM, for a Memory to be "accessible" to a Processor means that, when a segment runs on that processor, it can make demands (both space and bandwidth) on that memory. These resource demands are specified by means of transfer functions -- one for each memory accessible to the processor. Likewise, when a Memory Class is defined, associated with it is a list of associated Processors. Thus, the association is specified twice: once processor-to-memory, and once memory-to-processor. The consistency of this dual specification is validated by the Verify command.

As noted above, connectivities are specified using variable names as place holders. That is, a list of variable names is created. When a processor of the Class is instantiated, the variable names will be replaced with the instantiated names of "real" memories. The purpose of this command is to edit the list of variable names that has already been created (by use of the "Add Accessible Memory" command).

The Verify command will validate that the Memory Class name associated with each variable is, in fact, the name of a Memory Class as entered via the PERM Memory Class Create or Edit Header commands. This is primarily a question of correct spelling. Also, it will be verified that the variable names are all unique within a given Class; duplicate variable names are not permitted, although PERM does not detect them at this point in the data entry process. Thus, use distinct variable names in the Accessible Memories list for each Processor Class.

When the command is executed, the user is first presented with a menu of Processor Classes; the <Home> and <End> keys are used to scroll, and <Return> to select. Then, the current list of variables associated with that Processor Class is presented; scrolling and selection operate in the same way. Once the class and variable names have been selected, a menu is presented with the current data -- both the variable name, and the Memory Class associated with the variable.

NOTE:

One important use for this command is to correct spelling errors in the Memory Class field.

The user can move freely between the fields using the <arrow> keys. The variable name can be up to 32 characters, including special characters. The name entered for the Memory Class should agree exactly (that is, character by character) with a Memory Class name as entered using the Create Memory Class command or the Edit Memory Class Header command. Failure to do so will be detected by the Verify command.

The command can be exited at any time using the <ESC> key without altering the PE data structure. Once <Return> is entered, the permanent PE structure is altered to contain whatever edits the user has entered.

Delete an Accessible Memory Variable (@@PEPBG)

The purpose of this command is to delete a name from the list of variables designating the accessible memories associated with a Processor Class. These variables were previously created using the "Add Accessible Memory" command (or, edited using the "Edit Accessible Memory" command).

When the command is executed, the user is first presented with a menu of Processor Classes; the <Home> and <End> keys are used to scroll, and <Return> to select. Then, the current list of variables associated with that Processor Class is presented; scrolling and selection operate in the same way. Selection of the variable immediately results in its deletion from the current list of variables. The command can be exited without change to the PE data structure by entering <ESC>.

Add an Accessible Bus Variable (@@PEPBH)

The defining property of a Processor Class is its connectivity pattern -- that is, the list of other Processors, Memories, and Busses to which it is accessible. These connectivities are specified by means of variable names which act as place-holders until the Class is instantiated. When instantiated, the variable names are replaced by "real" names -- that is, names of other instantiated Processors, Memories, and Busses.

In PERM, for a Bus to be "accessible" to a Processor means that, when a segment runs on that processor, it can make bandwidth demands on that bus. These resource demands are specified by means of transfer functions -- one for each bus accessible to the processor. Likewise, when a Bus Class is defined, associated with it is a list of associated Processors. Thus, the association is specified twice: once processor-to-bus, and once bus-to-processor. The consistency of this dual specification is validated by the Verify command.

As noted above, connectivities are specified using variable names as place holders. That is, a list of variable names is created. When a processor of the Class is instantiated, the variable names will be replaced with the instantiated names of "real" busses. The purpose of this command is to create the list of variable names, and to associate with each variable name a Bus Class.

The Verify command will validate that the Bus Class name associated with each variable is, in fact, the name of a Bus Class as entered via the PERM Bus Class Create or Edit Header commands. This is primarily a question of correct spelling. Also, it will be verified that the variable names are all unique within a given Class; duplicate variable names are not permitted, although PERM does not detect them at this point in the data entry process. Thus, use distinct variable names in the Accessible Busses list for each Processor Class.

When the command is executed, the user is presented with a menu of Processor Classes; the <Home> and <End> keys are used to scroll, and <Return> to select. Then, a menu is offered, providing one field for the variable name, and one for the Bus Class the variable is to belong to. The user can move freely between the fields using the <arrow> keys. The variable name can be up to 32 characters, including special characters. The name entered for the Bus Class should agree exactly (that is, character by character) with a Bus Class name as entered using the Create Bus Class command or the Edit Bus Class Header command. Failure to do so will be detected by the Verify command.

The command can be exited at any time using the <ESC> key without altering the PE data structure. Once <Return> is entered, the permanent PE structure is altered to contain whatever edits the user has entered.

Edit an Accessible Bus Variable (@@PEPBI)

The defining property of a Processor Class is its connectivity pattern -- that is, the list of other Processors, Memories, and Busses to which it is accessible. These connectivities are specified by means of variable name which act as place-holders until the Class is instantiated. When instantiated, the variable names are replaced by "real" names -- that is, names of other instantiated Processors, Memories, and Busses.

In PERM, for a Bus to be "accessible" to a Processor means that, when a segment runs on that processor, it can make bandwidth demands on that bus. These resource demands are specified by means of transfer functions -- one for each bus accessible to the processor. Likewise, when a Bus Class is defined, associated with it is a list of associated Processors. Thus, the association is specified twice: once processor-to-bus, and once bus-to-processor. The consistency of this dual specification is validated by the Verify command.

As noted above, connectivities are specified using variable names as place holders. That is, a list of variable names is created. When a processor of the Class is instantiated, the variable names will be replaced with the instantiated names of "real" busses. The purpose of this command is to edit the list of variable names that has already been created (by use of the "Add Accessible Bus" command).

The Verify command will validate that the Bus Class name associated with each variable is, in fact, the name of a Bus Class as entered via the PERM Bus Class Create or Edit Header commands. This is primarily a question of correct spelling. Also, it will be verified that the variable names are all unique within a given Class; duplicate variable names are not permitted, although PERM does not detect them at this point in the data entry process. Thus, use distinct variable names in the Accessible Busses list for each Processor Class.

When the command is executed, the user is first presented with a menu of Processor Classes; the <Home> and <End> keys are used to scroll, and <Return> to select. Then, the current list of variables associated with that Processor Class is presented; scrolling and selection operate in the same way. Once the class and variable names have been selected, a menu is presented with the current data -- both the variable name, and the Bus Class associated with the variable.

NOTE:

> One important use for this command is to correct spelling errors in the Bus Class field.

The user can move freely between the fields using the <arrow> keys. The variable name can be up to 32 characters, including special characters. The name entered for the Bus Class should agree exactly (that is, character by character) with a Bus Class name as entered using the Create Bus Class command or the Edit Bus Class Header command. Failure to do so will be detected by the Verify command.

The command can be exited at any time using the <ESC> key without altering the PE data structure. Once <Return> is entered, the permanent PE structure is altered to contain whatever edits the user has entered.

Delete an Accessible Bus Variable (@@PEPBJ)

The purpose of this command is to delete a name from the list of variables designating the accessible busses associated with a Processor Class. These variables were previously created using the "Add Accessible Bus" command (or, edited using the "Edit Accessible Bus" command). When the command is executed, the user is first presented with a menu of Processor Classes; the <Home> and <End> keys are used to scroll, and <Return> to select. Then, the current list of variables associated with that Processor Class is presented; scrolling and selection operate in the same way. Selection of the variable immediately results in its deletion from the current list of variables. The command can be exited without change to the PE data structure by entering <ESC>.

Edit the Header of a Memory Class (@@PEMBA)

This command allows the user to change header information associated with an existing Memory Class -- in particular, its name, and the memory size and bandwidth. The user chooses the class to be edited by scrolling using the <Home> and <End> keys; when the cursor reaches the desired Class, enter <Return>, and the header will be displayed for editing. The command can be exited at any time without change to the PE data structure by pressing <ESC>.

The fields in the header can be selected using the <arrow> keys; use of <Return> terminates the command, and records any edits in the permanent PE data structure.

Since any instantiations of a Class inherit all the Class characteristics, changing the bandwidth and size parameters in the header automatically changes them in the associated instantiations. We recall that size is to be entered as an integer, and has units of Bytes. Bandwidth is also entered as an integer, and has units Bytes/Second. PERM will detect mis-formatted entries, and will prompt for re-entry.

Changes to the Class name should be used with caution, since connectivities associated with Classes (Processor, Memory, and Bus) are entered by specifying Class names. If the Class name is changed, references to it in connectivity specifications must also be changed, or the PE structure will fail verification.

Add an Accessible Processor Variable (@@PEMBB)

The defining property of a Memory Class is its connectivity pattern -- that is, the list of Processors which can access it. These connectivities are specified by means of variable names which act as place-holders until the Class is instantiated. When instantiated, the variable names are replaced by "real" names -- that is, names of instantiated Processors. Segments running on those processors can then, by means of transfer functions, make demands on the memories to which they are connected.

Associated with each Processor Class is a list of associated Memories. Similarly, with each Memory Class is a list of associated Processors. Thus, the association is specified twice: one processor-to-memory, and once memory-to-processor. The consistency of this dual specification is validated by the Verify command.

As noted above, connectivities are specified using variable names as place holders. That is, a list of variable names is created. When a memory of the Class is instantiated, the variable names will be replaced with the instantiated names of "real" processors. The purpose of this command is to create the list of variable names, and to associate with each variable name a Processor Class.

The Verify command will validate that the Processor Class name associated with each variable is, in fact, the name of a Processor Class as entered via the PERM Processor Class Create or Edit Header commands. This is primarily a question of correct spelling. Also, it will be verified that the variable names are all unique within a given Class; duplicate variable names are not permitted, although PERM does not detect them at this point in the data entry process. Thus, use distinct variable names in the Accessible Processors list for each Processor Class.

When the command is executed, the user is presented with a menu of Memory Classes; the <Home> and <End> keys are used to scroll, and <Return> to select. Then, a menu is offered, providing one field for the variable name, and one for the Processor Class the variable is to belong to. The user can move freely between the fields using the <arrow> keys. The variable name can be up to 32 characters, including special characters. The name entered for the Processor Class should agree exactly (that is, character by character) with a Processor Class name as entered using the Create Processor Class command or the Edit Processor Class Header command. Failure to do so will only be detected by the Verify command.

The command can be exited at any time using the <ESC> key without altering the PE data structure. Once <Return> is entered, the permanent PE structure is altered to contain whatever edits the user has entered.

Edit an Accessible Processor Variable (@@PEMBC)

The defining property of a Memory Class is its connectivity pattern -- that is, the list of Processors which can access it. These connectivities are specified by means of variable names which act as place-holders until the Class is instantiated. When instantiated, the variable names are replaced by "real" names -- that is, names of instantiated Processors. Segments running on those processors can then, by means of transfer functions, make demands on the memories to which they are connected.

Associated with each Processor Class is a list of associated Memories. Similarly, with each Memory Class is a list of associated Processors. Thus, the association is specified twice: one processor-to-memory, and once memory-to-processor. The consistency of this dual specification is validated by the Verify command.

As noted above, connectivities are specified using variable names as place holders. That is, a list of variable names is created. When a memory of the Class is instantiated, the variable names will be replaced with the instantiated names of "real" processors. The purpose of this command is to edit the list of variable names and associated Processor Classes that (as created by the "Add Accessible Processor" command).

The Verify command will validate that the Processor Class name associated with each variable is, in fact, the name of a Processor Class as entered via the PERM Processor Class Create or Edit Header commands. This is primarily a question of correct spelling. Also, it will be verified that the variable names are all unique within a given Class; duplicate variable names are not permitted, although PERM does not detect them at this point in the data entry process. Thus, use distinct variable names in the Accessible Processors list for each Processor Class.

When the command is executed, the user is first presented with a menu of Memory Classes; the <Home> and <End> keys are used to scroll, and <Return> to select. Then, the current list of variable names is presented, with scrolling and selection operating in the same manner. At this point, the current data is displayed for editing. Two fields are offered; one, for the variable names; and the other, for the associated Processor Class.

NOTE:

An important use for this command is to correct spelling errors in the Processor Class name. These errors will be pointed out by the Verify command.

The user can move freely between the fields using the <arrow> keys. The variable name can be up to 32 characters, including special characters. The name entered for the Processor Class should agree exactly (that is, character by character) with a Processor Class name as entered using the Create Processor Class command or the Edit Processor Class Header command. Failure to do so will be detected by the Verify command.

The command can be exited at any time using the <ESC> key without altering the PE data structure. Once <Return> is entered, the permanent PE structure is altered to contain whatever edits the user has entered.

Delete an Accessible Processor Variable (@@PEMBD)

The purpose of this command is to delete a name from the list of variables designating the accessible processors associated with a Memory Class. These variables were previously created using the "Add Accessible Processor" command (or, edited using the "Edit Accessible Processor" command).

When the command is executed, the user is first presented with a menu of Memory Classes; the <Home> and <End> keys are used to scroll, and <Return> to select. Then, the current list of variables associated with that Memory Class is presented; scrolling and selection operate in the same way. Selection of the variable immediately results in its deletion from the current list of variables. The command can be exited without change to the PE data structure by entering <ESC>.

Edit the Header of a Bus Class (@@PEBBA)

This command allows the user to change header information associated with an existing Bus Class -- in particular, its name and bandwidth. The user chooses the class to be edited by scrolling using the <Home> and <End> keys; when the cursor reaches the desired Class, enter <Return>, and the header will be displayed for editing. The command can be exited at any time without change to the PE data structure by pressing <ESC>.

The fields in the header can be selected using the <arrow> keys; use of <Return> terminates the command, and records any edits in the perma .nt PE data structure.
Since any instantiations of a Class inherit all the Class characteristics, changing the bandwidth parameter in the header automatically changes it in the associated instantiations. We recall that bandwidth is entered as an integer, and has units Bytes/Second. PERM will detect mis-formatted entries, and will prompt for re-entry.

Changes to the Class name should be used with caution, since connectivities associated with Classes (Processor, Memory, and Bus) are entered by specifying Class names. If the Class name is changed, references to it in connectivity specifications must also be changed, or the PE structure will fail verification.

Add an Accessible Processor Variable (@@PEBBB)

The defining property of a Bus Class is its connectivity pattern -- that is, the list of Processors which can access it. These connectivities are specified by means of variable names which act as place-holders until the Class is instantiated. When instantiated, the variable names are replaced by "real" names -- that is, names of instantiated Processors. Segments running on those processors can then, by means of transfer functions, make demands on the busses to which they are connected.

Associated with each Pro or Class is a list of associated Busses. Similarly, with each Bus Class is a list of associated Processors. Thus, the association is specified twice: one processor-to-bus, and once bus-to-processor. The consistency of this dual specification is validated by the Verify command.

As noted above, connectivities are specified using variable names as place holders. That is, a list of variable names is created. When a bus of the Class is instantiated, the variable names will be replaced with the instantiated names of "real" processors. The purpose of this command is to create the list of variable names, and to associate with each variable name a Processor Class.

The Verify command will validate that the Processor Class name associated with each variable is, in fact, the name of a Processor Class as entered via the PERM Processor Class Create or Edit Header commands. This is primarily a question of correct spelling. Also, it will be verified that the variable names are all unique within a given Class; duplicate variable names are not permitted, although PERM does not detect them at this point in the data entry process. Thus, use distinct variable names in the Accessible Processors list for each Processor Class.

When the command is executed, the user is presented with a menu of Bus Classes; the <Home> and <End> keys are used to scroll, and <Return> to select. Then, a menu is offered, providing one field for the variable name, and one for the Processor Class the variable is to belong to. The user can move freely between the fields using the <arrow> keys. The variable name can be up to 32 characters, including special characters. The name entered for the Processor Class should agree exactly (that is, character by character) with a Processor Class name as entered using the Create Processor Class command or the Edit Processor Class Header command. Failure to do so will only be detected by the Verify command.

The command can be exited at any time using the <ESC> key without altering the PE data structure. Once <Return> is entered, the permanent PE structure is altered to contain whatever edits the user has entered.

Edit an Accessible Processor Variable (@@PEBBC)

The defining property of a Bus Class is its connectivity pattern -- that is, the list of Processors which can access it. These connectivities are specified by means of variable names which act as place-holders until the Class is instantiated. When instantiated, the variable names are replaced by "real" names -- that is, names of instantiated Processors. Segments running on those processors can then, by means of transfer functions, make demands on the busses to which they are connected.

Associated with each Processor Class is a list of associated Busses. Similarly, with each Bus Class is a list of associated Processors. Thus, the association is specified twice: one processor-to-bus, and once bus-to-processor. The consistency of this dual specification is validated by the Verify command.

As noted above, connectivities are specified using variable names as place holders. That is, a list of variable names is created. When a bus of the Class is instantiated, the variable names will be replaced with the instantiated names of "real" processors. The purpose of this command is to edit the list of variable names and associated Processor Classes (as created by the "Add Accessible Processor" command).

The Verify command will validate that the Processor Class name associated with each variable is, in fact, the name of a Processor Class as entered via the PERM Processor Class Create or Edit Header commands. This is primarily a question of correct spelling. Also, it will be verified that the variable names are all unique within a given Class; duplicate variable names are not permitted, although PERM does not detect them at this point in the data entry process. Thus, use *distinct variable names in the Accessible Processors list for each Processor Class.*

When the command is executed, the user is presented with a menu of Bus Classes; the <Home> and <End> keys are used to scroll, and <Return> to select. Then, a menu of the current variable names is presented; scrolling and selection operate in the same manner. Finally, an editable header is shown containing the current data associated with the variable: that variable name; and, the Processor Class associated with the variable.

NOTE:

> An important use for this function is to correct spelling errors for the Processor Class name. The presence of such errors will be detected by the Verify command.

The user can move freely between the fields using the <arrow> keys. The variable name can be up to 32 characters, including special characters. The name entered for the Processor Class should agree exactly (that is, character by character) with a Processor Class name as entered using the Create Processor Class command or the Edit Processor Class Header command. Failure to do so will only be detected by the Verify command.

The command can be exited at any time using the <ESC> key without altering the PE data structure. Once <Return> is entered, the permanent PE structure is altered to contain whatever edits the user has entered.

Delete an Accessible Processor Variable (@@PEBBD)

The purpose of this command is to delete a name from the list of variables designating the accessible processors associated with a Bus Class. These variables were previously created using the "Add Accessible Processor" command (or, edited using the "Edit Accessible Processor" command).

When the command is executed, the user is first presented with a menu of Bus Classes; the <Home> and <End> keys are used to scroll, and <Return> to select. Then, the current list of variables associated with that Bus Class is presented; scrolling and selection operate in the same way. Selection of the variable immediately results in its deletion from the current list of variables. The command can be exited without change to the PE data structure by entering <ESC>.

Edit the Header of a Processor Instantiation (@@ PEPFA)

This command allows the user to change the name of an instantiation -- for example, because he discovers it has been misspelled. The user is prompted of the Class and name of the instantiation to be edited, using scrolling menus (<Home> and <End> to move the cursor, and <Return> to select). The Instantiation Header is then presented, and the instantiation name field can be edited. The <ESC> key exits the command without effecting the data structure, and the <Return> key executes the command and updates the structure.

Instantiate an Accessible Processor Variable (@@PEPFB)

The major defining characteristic of a Processor Class is the list of variables that specifies the connectivity of any processor in that Class is other Processors, Memories, and Busses. When a Processor Class is then instantiated, these connectivity variables are replaced by the names of real, instantiated Processors, Busses, and Memories. The purpose of this command is to enable the user to specify this assignment for the variables relating to Accessible Processors.

The user will be prompted for three pieces of information: (1) the Class of the processor whose instantiation is being edited; (2) the name of the instantiated processor (of that class) being edited; and (3) the variable name for one of the Accessible Processor variables whose value is to be specified. These selections are made successively from scrolling menus in the usual manner -- <Home> and <End> to move the cursor, and <Return> to select.

Next, a menu is presented with the field to be edited. The user is to enter the name of an instantiated processor, of the Class belonging to the variable. During verification, this name will be checked to validate that it is, indeed, the name of an instantiated processor belonging to the Processor Class associated with the variable. It will also be verified that the list of associated processors for that processor contains the name of the processor being edited.

As usual, the user can exit this command without changing the PE data structure by using <ESC>. The <Return> key executes the command, and updates the PE data structure.

Instantiate an Accessible Memory Variable (@@PEPFC)

The major defining characteristic of a Processor Class is the list of variables that specifies the connectivity of any processor in that Class is other Processors, Memories, and Busses. When a Processor Class is then instantiated, these connectivity variables are replaced by the names of real, instantiated Processors, Busses, and Memories. The purpose of this command is to enable the user to specify this assignment for the variables relating to Accessible Memories.

The user will be prompted for three pieces of information: (1) the Class of the processor whose instantiation is being edited; (2) the name of the instantiated processor (of that class) being edited; and (3) the variable name for one of the Accessible Memory variables whose value is to be specified. These selections are made successively from scrolling menus in the usual manner -- <Home> and <End> to move the cursor, and <Return> to select.

Next, a menu is presented with the field to be edited. The user is to enter the name of an instantiated processor, of the Class belonging to the variable. During verification, this name will be checked to validate that it is, indeed, the name of an instantiated processor belonging to the Memory Class associated with the variable. It will also be verified that the list of associated processors for that processor contains the name of the processor being edited.

As usual, the user can exit this command without changing the PE data structure by using <ESC>. The <Return> key executes the command, and updates the PE data structure.

Instantiate an Accessible Bus Variable (@@PEPFD)

The major defining characteristic of a Processor Class is the list of variables that specifies the connectivity of any processor in that Class is other Processors, Memories, and Busses. When a Processor Class is then instantiated, these connectivity variables are replaced by the names of real, instantiated Processors, Busses, and Memories. The purpose of this command is to enable the user to specify this assignment for the variables relating to Accessible Busses.

The user will be prompted for three pieces of information: (1) the Class of the processor whose instantiation is being edited; (2) the name of the instantiated processor (of that class) being edited; and (3) the variable name for one of the Accessible Processor variables whose value is to be specified. These selections are made successively from scrolling menus in the usual manner -- <Home> and <End> to move the cursor, and <Return> to select.

Next, a menu is presented with the field to be edited. The user is to enter the name of an instantiated processor, of the Class belonging to the variable. During verification, this name will be checked to validate that it is, indeed, the name of an instantiated processor belonging to the Bus Class associated with the variable. It will also be verified that the list of associated processors for that processor contains the name of the processor being edited.

As usual, the user can exit this command without changing the PE data structure by using <ESC>. The <Return> key executes the command, and updates the PE data structure.

Edit the Header of a Memory Instantiation (@@PEMFA)

This command allows the user to change the name of an instantiation -- for example, because he discovers it has been misspelled. The user is prompted for the Class and name of the instantiation to be edited, using scrolling menus (<Home> and <End> to move the cursor, and <Return> to select). The Instantiation Header is then presented, and the instantiation name field can be edited. The <ESC> key exits the command without effecting the data structure, and the <Return> key executes the command and updates the structure.

Instantiate an Accessible Processor Variable (@@PEMFB)

The major defining characteristic of a Memory Class is the list of variables that specifies the connectivity of any memory in that Class to other Processors, Memories, and Busses. When a Memory Class is then instantiated, these connectivity variables are replaced by the names of real, instantiated Processors. The purpose of this command is to enable the user to specify this assignment for the Accessible Processor variables.

The user will be prompted for three pieces of information: (1) the Class of the memory whose instantiation is being edited; (2) the name of the instantiated memory (of that class) being edited; and (3) the variable name for one of the Accessible Processor variables whose value is to be specified. These selections are made successively from scrolling menus in the usual manner -- <Home> and <End> to move the cursor, and <Return> to select.

Next, a menu is presented with the field to be edited. The user is to enter the name of an instantiated processor, of the Class belonging to the variable. During verification, this name will be checked to validate that it is, indeed, the name of an instantiated processor belonging to the Processor Class associated with the variable. It will also be verified that the list of associated memories for that processor contains the name of the memory being edited.

As usual, the user can exit this command without changing the PE data structure by using <ESC>. The <Return> key executes the command, and updates the PE data structure.

Edit the Header of a Bus Instantiation (@@PEBFA)

This command allows the user to change the name of an instantiation -- for example, because he discovers it has been misspelled. The user is prompted for the Class and name of the instantiation to be edited, using scrolling menus (<Home> and <End> to move the cursor, and <Return> to select). The Instantiation Header is then presented, and the instantiation name field can be edited. The <ESC> key exits the command without effecting the data structure, and the <Return> key executes the command and updates the structure.

Instantiate an Accessible Processor Variable (@@PEBFB)

The major defining characteristic of a Bus Class is the list of variables that specifies the connectivity of any memory in that Class to other Processors. When a Bus Class is then instantiated, these connectivity variables are replaced by the names of real, instantiated Processors. The purpose of this command is to enable the user to specify this assignment for the Accessible Processor variables.

The user will be prompted for three pieces of information: (1) the Class of the memory whose instantiation is being edited; (2) the name of the instantiated memory (of that class) being edited; and (3) the variable name for one of the Accessible Processor variables whose value is to be specified. These selections are made successively from scrolling menus in the usual manner -- <Home> and <End> to move the cursor, and <Return> to select.

Next, a menu is presented with the field to be edited. The user is to enter the name of an instantiated processor, of the Class belonging to the variable. During verification, this name will be checked to validate that it is, indeed, the name of an instantiated processor belonging to the Processor Class associated with the variable. It will also be verified that the list of associated busses for that processor contains the name of the bus being edited.

As usual, the user can exit this command without changing the PE data structure by using <ESC>. The <Return> key executes the command, and updates the PE data structure.

2.3.1.2.2 Help Files for Task Class Operations

Task Class Operations (@@T)

This top-level command opens onto a tree of sub-menus that allow the user to operate on (i.e., save, load, create, edit, delete) Task Classes, including particularly Threads and Segments.

NOTE:

The Task Class operations require that a validated Processor Ensemble (PE) data structure be currently in memory, and Task Classes are created with reference to that PE structure. The PE structure is obtained by using the appropriate commands in the PE command menu (Create or Load).
When the Task Class menu is entered, one of two states exists:

(1) there are no Task Classes currently in memory; or,
(2) there are some Task Classes currently in memory.

If (1) holds, the user's first order of business will be to get one or more Task Classes into memory -- either by creating them using the Create command, or by loading them into memory using the Load command. If, on the other hand, Task Classes do exist in memory, then the Task Class operations can be issued without further ado.

The PERM commands are primarily intended for data entry. The user should already have organized the Task Class segments and threads prior to using PERM, including the adoption of mnemonic naming conventions.

Create a Task Class (@@TC)

Unlike Processor Ensemble operations, PERM can hold multiple distinct Task Classes in memory concurrently. Thus, even if some Task Classes have already been Created or Loaded, this command can still be executed to create additional Task Classes. Task Classes created and verified using Task Class operations can then be instantiated into a System Load using Load operations (top level menu).

When this command is executed, the user will be presented with a menu, including two highlighted fields for editing. The first is the name of the Task Class. Task Class names for a given PE structure must all be unique, and PERM will not permit creation of identically named Task Classes. Task Class names can be up to 32 characters, including special characters. Also a field (optional) is provided for the name of the author of the Task Class; again, a 32-character limit is imposed. Additional information that is displayed (but not editable) includes the creation date, the associated PE structure, and the most recent verification date (null when Create is executed).

Once Task Classes have been created using this command, they can be edited (to add Segments and Threads).

Select a Target Task Class (@@TT)

All commands at this level (except Create) operate on only one Task Class, and hence will require the user to select that Task Class by scrolling through a menu presented for that purpose. The Target command allows the user to move (the name of) a particular Task Class to the top of the scrolling menu list. This will save time when a number of repetitive operations involving the same Task Class must be performed.

When the command is executed, a menu is presented displaying the current list of Task Classes. The user can move the cursor up and down (respectively) through this menu by using the <Home> and <End> keys (respectively). Entering <Return> moves the currently selected Task Class to the top of the scrolling stack. Like all other PERM commands, this one can be exited using <ESC> without changing the current TC structure in any way.

Edit a Task Class (@@TE)

This command opens onto a series of sub-menus to allow the user to edit and display all aspects of a Task Class, including Segments and Threads. The actual Task Class to be edited is not selected until the desired operation (e.g., create segment) is selected from the next sub-menu.

Remove a Task Class (@@TR)

The purpose of this command is to delete a Task Class data structure from memory. The user will be presented with a menu; the Task Class is then selected by scrolling up (<Home>) or down (<End>) till the desired Task Class is highlighted; pressing <Return> then causes the selected TC structure to be deleted from memory. No "confirm" prompt is presented. The command can be exited using <ESC> without change to the PERM TC structures.

NOTE:

> Once this command is exercised against a selected Task Class, all information contained in that Task Class is immediately and irrevocably lost. Hence, if there is any chance that this information may be subsequently needed, it is best to first save the information to permanent disk by using the Save command. Then, if desired, the data structure can be re-loaded into memory using the Load command.

Display a Task Class Data Structure (@@TD)

The purpose of this command is to print to screen the complete current data structure associated with a selected Task Class. When the command is executed, the user selects the desired Task Class by scrolling up (<Home>) or down (<End>), and then pressing <Return> when the desired Task Class is highlighted. The data structure will then be printed to the screen, in a manner similar to the DOS "more" command. However, scrolling through the display can be terminated at any time by entering <c>.

The format of the displayed information is as follows:

1. Task Class Header Information
    - [Name, Processor Ensemble, Author, Creation date, Validation Status, and most recent validation date and time.]
2. Input and Output Dependencies Variables
    - For each Input Dependency Variable
    - Name and Task Class

   For each Output Dependency Variable
    - Name and Task Class
3. Segment Table
    - For Each Segment Class
        - Segment Name Class
        - Processor Class
        - Number of Instantiations
        - Segment Class Type (Join, OS, AC)
        - Transfer Function Coefficients
            - Run Time
            - Memory Size (one for each accessible memory variable)
            - Memory Bandwidth (one for each accessible memory variable)
            - Bus Bandwidth (one for each accessible bus variable)
4. Thread Table
    - For each Thread
        - Thread Name
        - Processor Name (instantiated) and Class
        - Segment List
            - For each Segment
                - Segment Name
                - Segment Class
                - Segment Type (Join, OS, AC)
                - For Join Segments,
                    - Predecessor Thread and Segment

Another way to review the data for a Task Class is to use the Print command. This command will print to a user-selected DOS file an identical ASCII copy of the data sent to the terminal by the Display command. That file can then be operated on by any DOS program or utility (i.e., the DOS "print" command, a text editor, etc.). However, this will require first exiting PERM. Thus, the advantage of the Display command is that it allows the user to see the data structure without having to exit PERM.

Load a Task Class From Disk (@@TL)

The purpose of this command is to enable the user to restore to memory a Task Class structure that has previously been saved to disk using the Save command. The user will be prompted for the file name. A full DOS path (including drive and directory) can be entered; if this is omitted, PERM will use the default directory (the directory from which PERM was executed). If PERM cannot find or open the named file, it reports back an error message, and waits for further instruction; and similarly, if it finds that the file does not contain a TC data structure, or if it finds that the name of the TC data structure in the file duplicates the name of an existing TC data structure.

NOTE:

> PERM requires correspondence between a Task Class and a PE data structure. When a Task Class is created, that correspondence is est?blished automatically; the code inserts the name of the current PE data structure into the Task Class header. However, it is possible to load a Task Class from disk whose associated Proc ssor Ensemble name does not agree with the name of the Processor Ensemble currently in memory. In this case, PERM changes the PE name to match the one currently in memory, and notifies the user of this fact. As the warning message says, however: if the Task Class structures (Segments, etc.) do not agree with the current PE structure, unpredictable system response can be expected, and the resulting melange will not pass verification. To summarize: be sure that the Task Class you load into memory agrees with the PE structure currently in memory; PERM will attempt to detect inconsistencies, but this is ultimately the user's responsibility.

Save a Task Class to Disk (@@TS)

The purpose of this command is to enable a user to save (for later retrieval and reuse) the current status of a Task Class (TC) data structure to permanent disk. When executed, the user will select the Task Class to be saved by scrolling up (<Home>) or down (<End>) through the displayed list until the desired Task Class is selected (<Return>). The user will also be prompted for a DOS file name; this can include a full path name (drive and directory), or PERM will use the default directory (the directory from which PERM executed).

If the file already exists, it will be overwritten by PERM, and its existing contents lost. If PERM cannot open the file, an error message is displayed, and PERM waits for further instruction.

Once the TC structure has been saved using this command, it can later be recalled into memory using the Load command. Also, it is not necessary that the saved TC structure be complete or verified; partial and/or incomplete structures can be saved, and then later recalled for completion or editing. As a general rule, *structures should be regularly saved during an editing* session so that work will not be lost due to system failure (power outage, etc.).

NOTE:

> There is an important distinction between a file created using the Save command and one created using the Print command. A file created using the Save command is in the internal, binary format required by PERM software. The Print, command, on the other hand, creates a user-readable, ASCII file which can be edited, printed, etc. using DOS utilities and programs. Such a file is useful in that it enables a user to view the contents of the data structure, but it cannot be used by PERM directly to restore a TC structure. Using DOS extensions (e.g., ".DAT" and ".TXT"), the user can create a mnemonic naming convention to help enforce the important distinction.

Print an ASCII File of a Task Class (@@TP)

The purpose of this command is to print to a permanent disk file an ASCII file of the contents of a user-selected Task Class. The format of the data is as follows:

1. Task Class Header Information
   [Name, Processor Ensemble, Author, Creation date, Validation Status, and most recent validation date and time.]
2. Input and Output Dependencies Variables
   - For each Input Dependency Variable
     - Name and Task Class
   - For each Output Dependency Variable
     - Name and Task Class
3. Segment Table
   - For Each Segment Class
     - Segment Name Class
     - Processor Class
     - Number of Instantiations
     - Segment Class Type (Join, OS, AC)
     - Transfer Function Coefficients
       - Run Time
       - Memory Size (one for each accessible memory variable)
       - Memory Bandwidth (one for each accessible memory variable)
       - Bus Bandwidth (one for each accessible bus variable)
4. Thread Table
   - *For each Thread*
     - Thread Name
     - Processor Name (instantiated) and Class
     - Segment List
       - For each Segment
         - Segment Name
         - Segment Class
         - Segment Type (Join, OS, AC)
         - For Join Segments,
           - Predecessor Thread and Segment

When executed, the user will select the Task Class to be printed by scrolling up (<Home>) or down (<End>) through the displayed list until the desired Task Class is selected (<Return>). The user will also be prompted for a DOS file name; this can include a full path name (drive and directory), or PERM will use the default directory (the directory from which PERM executed).

If the file already exists, it will be overwritten by PERM, and its existing contents lost. If PERM cannot open the file, an error message is displayed, and PERM waits for further instruction.

NOTE:

> There is an important distinction between a file created using the Save command and one created using the Print command. A file created using the Save command is in the internal, binary format required by PERM software. The Print, command, on the other hand, creates a user-readable, ASCII file which can be edited, printed, etc. using DOS utilities and programs. Such a file is useful in that it enables a user to view the contents of the data structure, but it cannot be used by PERM directly to restore a TC structure. Using DOS extensions (e.g., ".DAT" and ".TXT"), the user can create a mnemonic naming convention to help enforce the important distinction.

Verify a Task Class (@@TV)

The purpose of this function is to verify the consistency of a Task Class data structure, both internally, and against its associated (verified) Processor Ensemble data structure. The verification function checks for many things. Any inconsistencies are reported back to the user, with an error message explaining the exact place where the error was found, and the exact type of the error. Using this information, the user can use the PERM TC edit tools to correct the errors.

NOTE:

> The user may wish to use the PERM Log facilities (top-level commands) if the error list is too long to remember easily. If a Log file is open, PERM will write the error message not only to the screen, but also to the Log file. That file can then be accessed (via DOS programs and utilities), for example, to obtain a hard-copy.

Only verified Task Classes can be used to create system loads, the basic data structure input to the Compute component of PERM.

Edit the Header of a Task Class (@@TEA)

This function permits the user to see and edit the header information of a currently memory-resident Task Class. First, the list of currently active Task Classes is displayed, and the user selects the desired one by moving the cursor up <Home> or down <End>, and then <Return> to select. The header menu is then presented, and the "name" and "author" fields are available for editing. Other system information (time of last verify, etc.) is also displayed, but cannot be edited.

NOTE:

Task Class names should all be unique. Do not rename an existing Task Class to agree with the name of another Task Class. If you do so, results are indeterminate, and the Task Class structure will not pass a Load Verify.

Edit the Dependency Variables List for a Task Class (@TEB)

This function opens onto a sub-menu of functions to create, view, edit, and delete the dependency variables (forward and backward) of a Task Class. These functions are individually explained under their own headings. Generally, however, a Task will have both predecessors and successors in the directed graph of Tasks that constitutes a Load. The potential for such dependencies is built into the Task Class using variable names that are created and edited by these functions. When a Task Class is instantiated during Load Definition, these variable names are then replaced by the names of other instantiated Tasks -- in the same manner as Classes are instantiated in Processor Ensemble definition.

Only the dependency variables of currently memory-resident Task Classes can be operated on by these functions. Further, these Task Classes should agree with the memory-resident (and verified) Processor Ensemble data structure.

Display the Dependency Variables for a Task Class (@@TEC)

This function prints to the screen a textual, ASCII listing of the current dependency variables list for a Task Class. The user will first be prompted to select which of the currently memory-resident Task Classes should be displayed (<Home> and <End> to move the cursor through the list, and <Return> to select). The predecessor and successor variables, and their associated Task Classes, are then scrolled to the screen, with the <c> key available to stop further output if desired.

Create a Segment Class (for a Task Class) (@@TED)

A Task Class consists of several structures: the input and output dependency variables, a list of segment classes, and a list of Threads created from those segment classes. In particular, Threads can only be created from Segment Classes associated with the Task Class. If it is desired to use a Segment definition in different Task Classes (say, because the Segment represents a commonly used block of code, such as a sorting routine), the Segment must be re-created in both Task Classes. Note that the "Import Segment Class" function makes this easy: once a Segment Class has been created in one Task Class, its definition can be duplicated in other Task Classes.

There are several fields to be entered when a Segment Class is defined. These include: (1) the Name of the Segment Class; (2) the Processor Class associated with it; (3) the Type of segment (Application Code, Operating System, or Join); and (4) the coefficients that define the Run Time for the segment as a function of Data Set size. Fields are provided for each of these, and we will discuss them separately.

The user will first be prompted to select the Task Class with which the Segment Class (to be created) is to be associated: <Home> and <End> move the cursor to the desired Task Class, and <Return> selects it.

Next, a menu containing several fields is presented. The first field is the Segment Class name. This should be chosen to generically describe the function performed by the segment, since the segment can be instantiated on several different threads. Up to 32 characters can be entered, including special characters. The Segment Class names within a Task Class should all be unique, but the same Segment Class name can be used in different Task Classes without violating the PERM verification rules. Pressing <Return> enters the name, and moves the cursor to the next field.

Next, the user will be prompted to identify the Type of Segment Class -- Application Code, Operating System, or Join. "Join" segments are used to specify inter-thread sequential dependencies. Operating System segments are accounted for separately in computing system overhead. The user uses <Home> and <End> to move the cursor to the desired field, and <Return> to select it.

Next, the user will be prompted to select a Processor Class for the Segment Class. That is, in building threads, only processors of the designated Processor Class will be available to "run" segments of this Segment Class. The Processor Classes are taken from the verified Processor Ensemble that must have been loaded before Task Class operations were begun. As usual, <Home> and <End> move the cursor to the desired point, and <Return> selects it.

Finally, six numerical fields are provided at the bottom to specify the Run Time Transfer Function. The meaning of these fields is explained on the menu, and their default values are shown. Note, in particular, that these must be selected to agree with whatever units are chosen to specify the data set size (number of bytes, number of tracks, number of records, etc.). PERM assumes that the output of a Run Time Transfer Function is in Seconds. PERM will only accept floating point values in the range 0.0 <= x <= 1.0 for the reduction factor; the remaining coefficients are entered in floating point format.

Even after a field has been edited, the user can return to it for changes by using the <Arrow> keys. Also, the menu itself can be exited at any time without modifying the Task Class data structure by entering <ESC>. The only way to exit the menu, and create a permanent Segment Class entry, is to move the cursor to the bottom-rightmost field (using <Arrow> keys or successive <Return>s) and press <Return>. This completes the function, and returns control to the Task Class Edit menu.

There are a number of other transfer functions that must be specified for a Segment Class -- one for each accessible Bus, and two for each Accessible Memory. These transfer functions are entered using the Edit Segment Class functions (which, see). Since those transfer function coefficients have default values, it is not necessary to explicitly edit them to pass Verification.

Edit a Segment Class (for a Task Class) (@@TEE)

The purpose of this function is to allow the user to edit both header information and Transfer Function information for a Segment Class. It immediately opens onto a submenu of available functions.

There are two primary purposes for this capability. The first is the most important -- the specification of Transfer Function coefficients for Accessible Busses and Memories. The commands that permit entering and editing those coefficients are accessed using this menu option. The second is to make changes to existing Segment Class definitions, especially those that may have been imported from other Task Classes (using the "Import Segment Class" capability).

Once a Segment Class has been instantiated on a Thread, the only editing that is permitted is to the values of its Transfer Function coefficients. That is, its name, Type, and associated Processor Class cannot be altered once a Segment belonging to the Class has been added to a Thread.

Delete a Segment Class (from a Task Class) (@@TEF)

This function allows a previously Created (or Imported) Segment Class to be deleted from a Task Class. This operation can only be performed if the Segment Class has not been instantiated as a Segment on a Thread. (In general, the only permitted editing of an instantiated Segment Class is to the values of its Transfer Function coefficients.)

The user will first be asked to select the Task Class from which the Segment Class is to be deleted (as usual, <Home> and <End> move the cursor, and <Return> selects). Then, the Segment Classes will be displayed, and the one to be deleted is selected in the same manner. The removal is immediate; no "confirm" is offered or required. The command can be exited at any time without change to the Task Class data structure by entering <ESC>.

Display a Segment Class (from a Task Class) (@@TEG)

This command prints an ASCII text description of a Segment Class belonging to a Task Class to the screen. Successive pages of the description are scrolled to the screen, and the output can be terminated at any point using the <c> key.

The information displayed is one entry from the "Segment Table" for the Task Class. It contains the following information:

Segment Class Name
Processor Class
Number of Instantiations
Segment Class Type (Join, OS, AC)
Transfer Function Coefficients
    Run Time
    Memory Size (one for each accessible memory variable)
    Memory Bandwidth (one for each accessible memory variable)
    Bus Bandwidth (one for each accessible bus variable)

The user will first be prompted to select the Task Class from the list of Task Classes in memory: <Home> and <End> move the cursor, and <Return> selects. Then, the list of Segment Classes for that Task Class will be presented, and the one to be displayed is selected in the same manner.

Import a Segment Class (From a Task Class, to a Task Class) (@@TEH)

A Task Class consists of several structures: the input and output dependency variables, a list of Segment Classes, and a list of Threads created from those Segment Classes. In particular, Threads can only be created from Segment Classes associated with the Task Class. If it is desired to use a Segment Class definition in different Task Classes (say, because the Segment represents a commonly used block of code, such as a sorting routine), the Segment must be re-created in both Task Classes. The "Import Segment Class" function makes this easy: once a Segment Class has been created in one Task Class, its definition can be duplicated in other Task Classes.

Three things must be specified to this function: (1) the Task Class from which the Segment Class is to be taken (the source); (2) the Task Class into which the Segment Class is to be placed (the target); and (3) the Segment Class to be copied. These three are selected from scrolling lists using (as usual) the <Home> and <End> keys to move the cursor, and the <Return> key to select. Once executed, the Segment Class is added to the list of Segment Classes associated with the Task Class, and can be edited and instantiated on Threads.

A possible use for this capability is to create a Task Class that consists only of Segment Classes -- no Threads. This can then be treated as a Library of Segment Classes to be used by other Task Classes that describe the execution Threads. This will ensure the consistency of Segment Class properties used across multiple Task Classes.

It is not permitted to Import a Segment Class whose name agrees with a Segment Class name already in the Task Class. This is consistent with the general PERM requirement for uniqueness of names within a PERM structural entity.

Create a Thread (for a Task Class) (@@TEI)

The purpose of this command is to Create the header information for a Thread in a Task Class. Four pieces of information are required: (1) the Task Class to which the Thread is to be added; (2) the Processor Class (from the Processor Ensemble) on which the Thread is to run; (3) the Instantiated Processor (belonging to that Processor Class) on which the Thread is to run; and (4) the name of the Thread. The first three of these are chosen from successively presented scrolling menus in the usual manner -- using <Home> and <End> keys to move the cursor, and <Return> to select. Note that at most one Thread can be associated with each instantiated processor; an attempt to create a second Thread will be detected as an error, and the user will be returned to the Edit Task Class menu.

The fourth piece of information -- the Thread name -- is then entered in the menu provided. Up to 32 characters are permitted, including special characters if desired. Thread names within a Task Class must be unique; duplicate names will not pass the Task Class verify function. Once a Thread has been created, it can be operated on using the Thread Editing functions provided.

Move a Thread to the Top of the Thread List (of a Task Class) (@@TEJ)

When editing Threads, a scrolling menu will be presented from which the Thread to be edited can be selected (using the <Home>, <End> and <Return> keys in the usual manner). This function moves the (name of the) chosen Thread to the top of the menu so that it can be selected immediately (using <Return>) without having to move the cursor at all. Thus, it behaves with regard to Thread names exactly as the Target command (from the top-level Task Class menu) behaves with regard to Task Class names.

The user will first select the Task Class (using the <Home> and <End> keys to move the cursor, and the <Return> key to select). Next, the currently active Threads belonging to that Task Class are presented, and the target Thread is selected in the same manner.

Use of this function (together with the Target command) can reduce the number of key strokes required in repetitive tasks involving the creation and editing of a Thread.

Edit a Thread (for a Task Class) (@@TEK)

The purpose of this command is enable the user to add Segments to a Thread (or, edit them, or delete them). It opens onto a sub-menu that offers the commands that effect this capability.

Delete a Thread (from a Task Class) (@@TEL)

This command enables a user to delete a Thread from a Task Class data structure. First, the Task Class is selected from a scrolling menu (in the usual way, using the <Home> and <End> keys to move the cursor, and <Return> to select). Then, the Thread to be deleted is chosen in the same manner.

Once <Return> is pressed, the selected Thread is removed immediately; no "confirm" is offered or required. However, the command can be exited at any time without changing the Task Class data structure by entering <ESC>.

Display the Threads (for a Task Class) (@@@TEM)

The command prints to the screen an ASCII display of the Threads associated with the selected Task Class. This includes the header information for each Thread as well as its list of Segments. The information displayed is as follows:

Task Class Name and Header Data
Thread Table
- For each Thread in the Task Class
  - Thread Name
  - Processor Name (instantiated) and Class
  - Segment List

For each Segment
Segment Name
Segment Class
Segment Type (Join, OS, AC)
For Join Segments,
Predecessor Thread and
Segment

The data is scrolled to the screen in pages, and the <c> key can be used to halt the display at any point.

Copy one Thread to Another (in a Task Class) (@@TEN)

Often, an identical (or nearly identical) software sequence is performed on multiple processors. In PERM terminology, this means that the Threads associated with those two processors are (nearly) identical -- that is, that the same sequence of Segment Classes constitutes the two Threads.

This command enables the user to copy the sequence of segments created for one Thread directly onto another Thread. The only condition is that the Target Thread (the thread onto which the Segment Sequence is to be copied) must not currently have any segments; if this is not already the case, it can be caused by using the "Clear Thread" command.

The user specifies three things: (1) the Task Class in which the "copy" operation is to be performed; (2) the Thread from which the Segments are to be copied (the Source); and (3) the Thread to which the Segments are to be copied (the Target). These are selected from a sequence of scrolling menus in the usual manner (<Home> and <End> to move the cursor, and <Return> to select).

The following restrictions apply and are enforced by PERM. First, the Source and Target must be associated with the same Processor Class. Second, the Target Thread must be Clear (that is, must not have any Segments on it). Third, the two Threads must belong to the same Task Class.

Once the Segments have been copied, they can be edited using the "Edit Thread" commands. For example, their instantiation names can be changed, they can be deleted, new Segments can be added or inserted, etc.

NOTE:

> The user will want to verify that instantiated JOIN Segments copied using this command retain their inter-thread dependencies. If not, they will have to be edited to reflect the altered dependencies.

It is not necessary to rename the Segments copied from one Thread to another in this way. Segment names may be identical across Threads (just as Segment Class names may be identical across Task Classes); it is only necessary that all segments within a Thread have unique names.

Clear a Thread (in a Task Class) (@@TEO)

This command causes all Segments associated with a Thread to be deleted. The Thread Header information remains; only the Segments are lost. This can be used, for example, to prepare a Thread to be the Target for a "Copy Thread" command.

The user selects both the Task Class and the Thread from scrolling menus in the usual manner (<Home> and <End> to move the cursor, and <Return> to select). The command can be exited at any time without altering the Task Class data structure by entering <ESC>.

NOTE:

> No "confirm" is offered or required; once entered, the command immediately deletes all Segments. Thus, the user is cautioned to judicious use of the Save command (at the top level Task Class menu) before extensive editing is employed.

Add an Input Dependency Variable (to a Task Class) (@@TEBA)

The input and output dependency variables serve as place-holders until the directed graph of Tasks is created during Load Definition. That is, a Task Class can have any number of predecessors (of specified Classes), as indicated by the Input Dependency Variables; and can have any number of successors (of specified Classes), as indicated by the Output Dependency Variables.

To create an Input variable, the user must specify three things: (1) the Task Class being edited; (2) the name of the variable; and (3) the name of the Task Class to which the variable belongs. When an instance of the Task Class (being edited) is created (during Load Definition), the variable will be replaced by an instance of the Task Class (to which the variable belongs).

NOTE:

> During Load Definition, PERM will permit an Input or Output Dependency Variable to be replaced by the string "dummy". This matches all Task Classes, and essentially means that there is no dependency for that particular instantiation and that particular variable.

The Task Class (being edited) is selected from a scrolling menu in the usual manner (<Home> and <End> keys move the cursor, and <Return> selects). The variable name and Task Class name are then entered as (up to) 32-long ASCII strings, including special characters, if desired.

NOTE:

> The names of Input Dependency Variables should all be unique within a Task Class. Failure to adhere to this will result in indeterminate behavior, and will not pass the Task Class Verify function. Also, the Task Class to which the variable belongs should be (by the time the Load Definition in executed) a Verified Task Class. PERM will permit the use of Task Class names that are not yet verified, or that are not even in memory. It will, however, note this fact to the user when Verify is implemented. However, in Load Definition, PERM is not so forgiving, and requires that the Task Classes associated with the variables be, in fact, verified Task Classes currently in memory.

Edit an Input Dependency Variable (for a Task Class) (@@TEBB)

The list of Input Dependency Variables created using the Create command can be edited -- both the names of these variables, and the Task Classes to which they belong. The user must specify two things: (1) the Task Class being edited; and (2) the Input Dependency Variable (of that Class) being edited. Both these are selected from scrolling menus in the usual manner (<Home> and <End> to move the cursor, and <Return> to select). The menu containing the variable name and its Task Class is then available for editing.

As usual, names can be up to 32 characters, including special characters. The Task Class name should be the exact name of a verified Task Class (at the time Load Definition occurs -- see below). The <Arrow> keys can be used to move the cursor to a particular character in the name for change by overtyping; they can also be used to switch between the two fields. The command can be exited at any time without changing the Task Class data structure by using the <ESC> key.

NOTE:

> During Load Definition, PERM will permit an Input or Output Dependency Variable to be replaced by the string "dummy". This matches all Task Classes, and essentially means that there is no dependency for that particular instantiation and that particular variable.
>
> The names of Input Dependency Variables should all be unique within a Task Class. Failure to adhere to this will result in indeterminate behavior, and will not pass the Task Class Verify function. Also, the Task Class to which the variable belongs should be (by the time the Load Definition in executed) a Verified Task Class. PERM will permit the use of Task Class names that are not yet verified, or that are not even in memory. It will, however, note this fact to the user when Verify is implemented. However, in Load Definition, PERM is not so forgiving, and requires that the Task Classes associated with the variables be, in fact, verified Task Classes currently in memory.

Delete an Input Dependency Variable (from a Task Class) (@@TEBC)

This command will delete an Input Dependency Variable from a Task Class. The user specifies two things: (1) the Task Class being edited; and (2) the Input Dependency Variable to be deleted. Both are selected from scrolling menus in the usual manner (using <Home> and <End> keys to move the cursor, and <Return> to select). No "confirm" is required or accepted; once the command is executed, the variable is at once deleted from the Task Class data structure. However, the command can be exited at any time without change to the data structure by using the <ESC> key.

Add an Output Dependency Variable (to a Task Class) (@@TEBD)

The input and output dependency variables serve as place-holders until the directed graph of Tasks is created during Load Definition. That is, a Task Class can have any number of predecessors (of specified Classes), as indicated by the Input Dependency Variables; and can have any number of successors (of specified Classes), as indicated by the Output Dependency Variables.

To create an Output variable, the user must specify three things: (1) the Task Class being edited; (2) the name of the variable; and (3) the name of the Task Class to which the variable belongs. When an instance of the Task Class (being edited) is created (during Load Definition), the variable will be replaced by an instance of the Task Class (to which the variable belongs).

NOTE:

> During Load Definition, PERM will permit an Input or Output Dependency Variable to be replaced by the string "dummy". This matches all Task Classes, and essentially means that there is no dependency for that particular instantiation and that particular variable.

The Task Class (being edited) is selected from a scrolling menu in the usual manner (<Home> and <End> keys move the cursor, and <Return> selects). The variable name and Task Class name are then entered as (up to) 32-long ASCII strings, including special characters, if desired.

NOTE:

> The names of Output Dependency Variables should all be unique within a Task Class. Failure to adhere to this will result in indeterminate behavior, and will not pass the Task Class Verify function. Also, the Task Class to which the variable belongs should be (by the time the Load Definition is executed) a Verified Task Class. PERM will permit the use of Task Class names that are not yet verified, or that are not even in memory. It will, however, note this fact to the user when Verify is implemented. However, in Load Definition, PERM is not so forgiving, and requires that the Task Classes associated with the variables be, in fact, verified Task Classes currently in memory.

Edit an Output Dependency Variable (for a Task Class) (@@TEBE)

The list of Output Dependency Variables created using the Create command can be edited -- both the names of these variables, and the Task Classes to which they belong. The user must specify two things: (1) the Task Class being edited; and (2) the Output Dependency Variable (of that Class) being edited. Both these are selected from scrolling menus in the usual manner (<Home> and <End> to move the cursor, and <Return> to select). The menu containing the variable name and its Task Class is then available for editing.

As usual, names can be up to 32 characters, including special characters. The Task Class name should be the exact name of a verified Task Class (at the time Load Definition occurs -- see below). The <Arrow> keys can be used to move the cursor to a particular character in the name for change by overtyping; they can also be used to switch between the two fields. The command can be exited at any time without changing the Task Class data structure by using the <ESC> key.

NOTE:

> During Load Definition, PERM will permit an Input or Output Dependency Variable to be replaced by the string "dummy". This matches all Task Classes, and essentially means that there is no dependency for that particular instantiation and that particular variable.
>
> The names of Output Dependency Variables should all be unique within a Task Class. Failure to adhere to this will result in indeterminate behavior, and will not pass the Task Class Verify function. Also, the Task Class to which the variable belongs should be (by the time the Load Definition in executed) a Verified Task Class. PERM will permit the use of Task Class names that are not yet verified, or that are not even in memory. It will, however, note this fact to the user when Verify is implemented. However, in Load Definition, PERM is not so forgiving, and requires that the Task Classes associated with the variables be, in fact, verified Task Classes currently in memory.

Delete an Output Dependency Variable (from a Task Class) (@@TEBF)

This command will delete an Output Dependency Variable from a Task Class. The user specifies two things: (1) the Task Class being edited; and (2) the Output Dependency Variable to be deleted. Both are selected from scrolling menus in the usual manner (using <Home> and <End> keys to move the cursor, and <Return> to select). No "confirm" is required or accepted; once the command is executed, the variable is at once deleted from the Task Class data structure. However, the command can be exited at any time without change to the data structure by using the <ESC> key.

Edit the Header of a Segment Class (for a Task Class) (@@TEEA)

This function enables the user to view and edit the Header information that was entered when the Segment Class was Created (or Imported). All four fields -- name, Type, Processor Class, and Run Time Transfer Function coefficients -- are editable, using the <Arrow> keys to move between fields and among characters within a field.

he user selects the Task Class and the Segment Class to be edited from scrolling menus in the usual manner (<Home> and <End> keys to move the cursor, and <Return> to select).

All the naming considerations that applied when the Segment Class was Created apply here, as well. The first field is the Segment Class name. This should be chosen to generically described the function performed by the segment, since the segment can be instantiated on several different threads. Up to 32 characters can be entered, including special characters. The Segment Class names within a Task Class should all be unique, but the same Segment Class name can be used in different Task Classes without violating the PERM verification rules. Pressing <Return> enters the name, and moves the cursor to the next field. The <Arrow> keys can be used to move between fields without altering their contents.

Next, the user will be prompted to identify the Type of Segment Class -- Application Code, Operating System, or Join. "Join" segments are used to specify inter-thread sequential dependencies. Operating System segments are accounted for separately in computing system overhead. The user uses <Home> and <End> to move the cursor to the desired field, and <Return> to select it.

Next, the user will be prompted to select a Processor Class for the Segment Class. That is, in building threads, only processors of the designated Processor Class will be available to "run" segments of this Segment Class. The Processor Classes are taken from the verified Processor Ensemble that must have been loaded before Task Class operations were begun. As usual, <Home> and <End> move the cursor to the desired point, and <Return> selects it.

NOTE:

> When a Segment Class is Created, Transfer Functions for all associated Bus and Memory Variables are also created, and default values are assigned. However, if a Segment Class is edited by changing the Processor Class to which it belongs, those Associated Variables (and their Transfer Functions) no longer have meaning. A similar situation arises if the Processor Class (belonging to the Processor Ensemble) is edited subsequent to the Creation of the Segment Class. In both these cases, no default values for the Transfer Functions are created; the user is require to explicitly edit them and confirm their new values -- even if these are 0, as will often be the case. To avoid this, it may be preferable in some instances to simply delete the Segment Class entirely and start over, using Create.

If any instances of a Segment Class have been created on Threads, the first three fields of the Segment Class cannot be edited -- only the Run Time Transfer Function coefficients. Thus, if it is desired (for example) to change the Type of the Segment Class from Application Code to Operating System, any Segments (on Threads) drawn from that Segment Class must be deleted from their threads (before the editing occurs), and then reinserted (after the editing is complete).

Finally, six numerical fields are provided at the bottom to specify the Run Time Transfer Function. The meaning of these fields is explained on the menu, and their default values are shown. Note, in particular, that these must be selected to agree with whatever units are chosen to specify the data set size (number of bytes, number of tracks, number of records, etc.). The output of the Run Time Transfer Function is assumed to be in Seconds. PERM will only accept floating point values in the range $0.0 <= x <= 1.0$ for the reduction factor; the remaining coefficients are entered in floating point format.

Add a Memory Size Transfer Function (for a Task Class) (@@TEEB)

NOTE:

Ordinarily, this command should never be used. All operations on Transfer Function coefficients can, in the usual course of things, be performed using the "Edit Memory Requirements Function," <C>. The circumstances under which this command might be of value are explained below, and they depend on a more-than-superficial understanding of how PERM represents the Task Class data structures internally. Before using this command, see if the function you want to implement is not available using <C>. If it is, use <C>; if not, return here, and continue reading.

When a Segment Class is Created, PERM automatically also creates Transfer Functions -- one for every Accessible Bus variable in the associated Processor Class, and two (one for size requirements, and one for I/O requirements) for each Accessible Memory variable. These transfer functions are created so as to automatically correctly agree with the Processor Class of the Segment in both number and "type" (where "type" means variable name and Class).

In the ordinary course of things, this association between the Segment Class and the Processor Class would not change, and neither would the properties of the Processor Class. Thus, the Transfer Functions originally created should remain internally consistent: their value might change, but the variable name and Class would not.

In can happen, however, that this correspondence between the "type" of the Transfer Functions and the associated Processor Class is broken. This can happen in two ways. First, the Processor Ensemble, itself, can be edited, and the variables list and their Classes changed. If this happens, the list of Transfer Functions will no long agree, in variable name and Class, with the Accessible Bus and Memory variables of the Processor Class.

Another way that the disagreement can arise is if the Processor Class of the Segment Class is, itself, edited (using the Edit Segment Header function). In this case, the list of Accessible Bus and Memory variables of the new Processor Class need no longer agree with the variables list of the original Processor Class.

What this function is intended to do is to enable a knowledgeable user to create his own list of Transfer Functions for each of the new variables. He can combine the "Add" function with the "Delete" function to take away Transfer Functions for which variables no longer exist, and to create Transfer Functions for variables that have been introduced, and that differ from the original list of Transfer Functions created at the time the Segment Class itself was created.
In order to pass verification, the resulting list of Transfer Functions must have the following properties:

(1) each transfer function is associated with the correct name and Class of some Accessible Bus or Memory variable from the Processor Class associated with the Segment Class.

(2) for every Accessible Bus variable in the Processor Class, there exists exactly one Bus Bandwidth Transfer Function with the correct variable name and the correct Bus Class.

(3) for every Accessible Memory variable in the Processor Class, there exist exactly two Transfer Functions -- one for Memory Requirements, and one for Memory Bandwidth -- each with the correct variable name and the correct Memory Class.

Condition (1) says that every Transfer Function is associated with a variable from the Segment's Processor Class. Conditions (2) and (3) say that every variable -- whether Bus or Memory -- has exactly the right number and "type" of associated Transfer Functions.

Thus, any Transfer Function that fails to satisfy (1) - (3) can be deleted (using the "Delete" commands: <D>, <G>, and <J>), and any that are missing can be added (using the "Add" commands: <B>, <E>, and <H>).

When an "Add" command is used, the user will be prompted to select the Task Class and the Segment Class to be edited, using scrolling menus in the usual manner (<Home> and <End> to move the cursor, and <Return> to select). Then three fields will be offered for editing. The first is the name of an Accessible Resource variable -- in this case, a memory variable. The second is the Class of that variable. To pass verification, these must agree with a variable name and Class from the associated Processor Class.

NOTE:

> It is at this point that the "Add" function, <B>, differs from the "Edit" function, <C>. The edit function will present another scrolling menu to select the accessible memory variable (and its Class) from the list created when the Segment Class itself was created. In the "Add" function, the user enters this information directly from the keyboard.

Third, the values of the Transfer Function coefficients must be entered. Their meaning is fully explained on the menu itself. For Memory Requirement, the resulting units should be in Bytes (as a function of Input Data Size).

As with all other PERM commands, this command can be exited at any time without disturbing the Task Class data structure by entering <ESC>.

Edit a Memory Requirements Transfer Function (for a Task Class) (@@TEEC)

When a Segment Class is Created, PERM automatically also creates Transfer Functions - - one for every Accessible Bus variable in the associated Processor Class, and two (one for size requirements, and one for I/O requirements) for each Accessible Memory variable. These transfer functions are created so as to automatically correctly agree with the Processor Class of the Segment in both number and "type" (where "type" means variable name and Class). Also, the Transfer Function coefficients are assigned default values, chosen so that, if left unedited, the Transfer Function effectively is null -- it will make no demands on PERM resources.

When an instance of a Segment Class is instantiated on a Thread, the Accessible Resource variables -- in this case, Accessible Memory variables -- are replaced with the "real", instantiated entities that are accessible to the "real", instantiated processor associated with the Thread. Then, as PERM runs in the computational phase, the Segments will make demands on these resources as specified by the Transfer Functions and the input data size.

Because the default values of the Transfer Functions effectively render the demands on the resource "null", the Transfer Function coefficients must be edited to reflect the actual behavior of the software block of code being modeled. The purpose of this command is to enable editing of Transfer Functions coefficients for Memory Requirements -- the amount of memory the Segment will require when it is executed.

When the command is executed, the user will be prompted for three pieces of information: (1) the Task Class being edited; (2) the Segment Class (within the Task Class) being edited; and (3) the name of the Accessible Memory variable (from the list specified in the Processor Class definition of the Processor Class associated with the Segment Class). In each case, selection is made from a scrolling menu in the usual manner (<Home> and <End> to move the cursor, and <Return> to select). A menu is then presented with an explanation of the six coefficients that specify the Memory Requirements Transfer Function, and the user edits each in turn (using the <Arrow> keys and <Return> to move between fields and enter data). The units for output by the Transfer Function are Bytes. The command is completed (and the Task Class data structure is updated) when the cursor in on the final field, and <Return> is entered. At any point, the user can exit the command without modification to the Task Class data structure by entering <ESC>.

Delete a Memory Size Transfer Function (for a Task Class) (@@TEED)

NOTE:

> Do not use this command to get rid of Transfer Functions that are known to be "null". The default values for all Transfer Functions are set so that the Function will have no effect during PERM execution. Deleting Transfer Functions from the list will ordinarily cause the Task Class to fail verification, since it is required that all Accessible Bus and Memory variables have Transfer Functions, even if they do not contribute to resource utilization.

Ordinarily, this command should never be used. All operations on Transfer Function coefficients can, in the usual course of things, be performed using the "Edit Memory Requirements Function," <C>. The circumstances under which this command might be of value are explained below, and they depend on a more-than-superficial understanding of how PERM represents the Task Class data structures internally. Before using this command, see if the function you want to implement is not available using <C>. If it is, use <C>; if not, return here, and continue reading.

When a Segment Class is Created, PERM automatically also creates Transfer Functions -- one for every Accessible Bus variable in the associated Processor Class, and two (one for size requirements, and one for I/O requirements) for each Accessible Memory variable. These transfer functions are created so as to automatically correctly agree with the Processor Class of the Segment in both number and "type" (where "type" means variable name and Class).

In the ordinary course of things, this association between the Segment Class and the Processor Class would not change, and neither would the properties of the Processor Class. Thus, the Transfer Functions originally created should remain internally consistent: their value might change, but the variable name and Class would not. Thus, there would be no need to Delete a Transfer Function -- even a "Null" one. To do so would be to doom the Task Class data structure to fail verification.

In can happen, however, that this correspondence between the "type" of the Transfer Functions and the associated Processor Class is broken. This can happen in two ways. First, the Processor Ensemble, itself, can be edited, and the variables list and their Classes changed. If this happens, the list of Transfer Functions will no long agree, in variable name and Class, with the Accessible Bus and Memory variables of the Processor Class.

Another way that the disagreement can arise is if the Processor Class of the Segment Class is, itself, edited (using the Edit Segment Header function). In this case, the list of Accessible Bus and Memory variables of the new Processor Class need no longer agree with the variables list of the original Processor Class.

What this function is intended to do is to enable a knowledgeable user to create his own list of Transfer Functions for each of the new variables. He can combine the "Add" function with the "Delete" function to take away Transfer Functions for which variables no longer exist, and to create Transfer Functions for variables that have been introduced, and that differ from the original list of Transfer Functions created at the time the Segment Class itself was created.

In order to pass verification, the resulting list of Transfer Functions must have the following properties:

(1) each transfer function is associated with the correct name and Class of some Accessible Bus or Memory variable from the Processor Class associated with the Segment Class.

(2) for every Accessible Bus variable in the Processor Class, there exists exactly one Bus Bandwidth Transfer Function with the correct variable name and the correct Bus Class.

(3) for every Accessible Memory variable in the Processor Class, there exist exactly two Transfer Functions -- one for Memory Requirements, and one for Memory Bandwidth -- each with the correct variable name and the correct Memory Class.

Condition (1) says that every Transfer Function is associated with a variable from the Segment's Processor Class. Conditions (2) and (3) say that every variable -- whether Bus or Memory -- has exactly the right number and "type" of associated Transfer Functions.
Thus, any Transfer Function that fails to satisfy (1) - (3) can be deleted (using the "Delete" commands: <D>, <G>, and <J>), and any that are missing can be added (using the "Add" commands: <B>, <E>, and <H>).

When a "Delete" command is used, the user will be prompted to select the Task Class, Segment Class, and Transfer Function (indicated by Accessible Memory variable) to be deleted, using scrolling menus in the usual manner (<Home> and <End> to move the cursor, and <Return> to select). The execution of the command immediately causes the Transfer Function to be removed from the Task Class data structure; no "confirm" is offered or required. However, the command can be exited at any time without modification to the Task Class by entering <ESC>.

Add a Memory I/O Transfer Function (for a Task Class) (@@TEEE)

NOTE:

Ordinarily, this command should never be used. All operations on Transfer Function coefficients can, in the usual course of things, be performed using the "Edit Memory Requirements Function," <F>. The circumstances under which this command might be of value are explained below, and they depend on a more-than-superficial understanding of how PERM represents the Task Class data structures internally. Before using this command, see if the function you want to implement is not available using <F>. If it is, use <F>; if not, return here, and continue reading.

When a Segment Class is Created, PERM automatically also creates Transfer Functions -- one for every Accessible Bus variable in the associated Processor Class, and two (one for size requirements, and one for I/O requirements) for each Accessible Memory variable. These transfer functions are created so as to automatically correctly agree with the Processor Class of the Segment in both number and "type" (where "type" means variable name and Class).

In the ordinary course of things, this association between the Segment Class and the Processor Class would not change, and *neither would the properties* of the Processor Class. Thus, the Transfer Functions originally created should remain internally consistent: their value might change, but the vanable name and Class would not.

In can happen, however, that this correspondence between the "type" of the Transfer Functions and the associated Processor Class is broken. This can happen in two ways. First, the Processor Ensemble, itself, can be edited, and the variables list and their Classes changed. If this happens, the list of Transfer Functions will no long agree, in variable name and Class, with the Accessible Bus and Memory variables of the Processor Class.

Another way that the disagreement can arise is if the Processor Class of the Segment Class is, itself, edited (using the Edit Segment Header function). In this case, the list of Accessible Bus and Memory variables of the new Processor Class need no longer agree with the variables list of the original Processor Class.

What this function is intended to do is to enable a knowledgeable user to create his own list of Transfer Functions for each of the new variables. He can combine the "Add" function with the "Delete" function to take away Transfer Functions for which variables no longer exist, and to create Transfer Functions for variables that have been introduced, and that differ from the original list of Transfer Functions created at the time the Segment Class itself was created.

In order to pass verification, the resulting list of Transfer Functions must have the following properties:

(1) each transfer function is associated with the correct name and Class of some Accessible Bus or Memory variable from the Processor Class associated with the Segment Class.

(2) for every Accessible Bus variable in the Processor Class, there exists exactly one Bus Bandwidth Transfer Function with the correct variable name and the correct Bus Class.

(3) for every Accessible Memory variable in the Processor Class, there exist exactly two Transfer Functions -- one for Memory Requirements, and one for Memory Bandwidth -- each with the correct variable name and the correct Memory Class.

Condition (1) says that every Transfer Function is associated with a variable from the Segment's Processor Class. Conditions (2) and (3) say that every variable -- whether Bus or Memory -- has exactly the right number and "type" of associated Transfer Functions.

Thus, any Transfer Function that fails to satisfy (1) - (3) can be deleted (using the "Delete" commands: <D>, <G>, and <J>), and any that are missing can be added (using the "Add" commands: <B>, <E>, and <H>).

When an "Add" command is used, the user will be prompted to select the Task Class and the Segment Class to be edited, using scrolling menus in the usual manner (<Home> and <End> to move the cursor, and <Return> to select). Then three fields will be offered for editing. The first is the name of an Accessible Resource variable -- in this case, a memory variable. The second is the Class of that variable. To pass verification, these must agree with a variable name and Class from the associated Processor Class.

NOTE:

It is at this point that the "Add" function, <H>, differs from the "Edit" function, <I>. The "Edit" function will present another scrolling menu to select the accessible memory variable (and its Class) from the list created when the Segment Class itself was created. In the "Add" function, the user enters this information directly from the keyboard.

Third, the values of the Transfer Function coefficients must be entered. Their meaning is fully explained on the menu itself. For Memory I/O, the resulting units should be in Bytes (as a function of Input Data Size).

As with all other PERM commands, this command can be exited at any time without disturbing the Task Class data structure by entering <ESC>.

Edit a Memory I/O Transfer Function (for a Task Class) (@@TEEF)

When a Segment Class is Created, PERM automatically also creates Transfer Functions -- one for every Accessible Bus variable in the associated Processor Class, and two (one for size requirements, and one for I/O requirements) for each Accessible Memory variable. These transfer functions are created so as to automatically correctly agree with the Processor Class of the Segment in both number and "type" (where "type" means variable name and Class). Also, the Transfer Function coefficients are assigned default values, chosen so that, if left unedited, the Transfer Function effectively is null -- it will make no demands on PERM resources.

When an instance of a Segment Class is instantiated on a Thread, the Accessible Resource variables -- in this case, Accessible Memory variables -- are replaced with the "real", instantiated entities that are accessible to the "real", instantiated processor associated with the Thread. Then, as PERM runs in the computational phase, the Segments will make demands on these resources as specified by the Transfer Functions and the input data size.

Because the default values of the Transfer Functions effectively render the demands on the resource "null", the Transfer Function coefficients must be edited to reflect the actual behavior of the software block of code being modeled. The purpose of this command is to enable editing of Transfer Functions coefficients for Memory I/O -- the number of Bytes of Memory to be transferred between the Processor and the Memory when the block of code is executed.

When the command is executed, the user will be prompted for three pieces of information: (1) the Task Class being edited; (2) the Segment Class (within the Task Class) being edited; and (3) the name of the Accessible Memory variable (from the list specified in the Processor Class definition of the Processor Class associated with the Segment Class). In each case, selection is made from a scrolling menu in the usual manner (<Home> and <End> to move the cursor, and

<Return> to select). A menu is then presented with an explanation of the six coefficients that specify the Memory I/O Transfer Function, and the user edits each in turn (using the keyboard, <Arrow> keys and <Return> to move between fields and enter data). The units for output by the Transfer Function are Bytes, and should include Reads and Writes for both data and instructions. The user should also take into account cache memory local to the processor; the presence of cache memory can reduce the number of references to main memory for many types of code. The command is completed (and the Task Class data structure is updated) when the cursor located on the final field, and <Return> is entered. At any point, the user can exit the command without modification to the Task Class data structure by entering <ESC>.

Delete a Memory I/O Transfer Function (for a Task Class) (@@TEEG)

NOTE:

> Do not use this command to get rid of Transfer Functions that are known to be "null". The default values for all Transfer Functions are set so that the Function will have no effect during PERM execution. Deleting Transfer Functions from the list will ordinarily cause the Task Class to fail verification, since it is required that all Accessible Bus and Memory variables have Transfer Functions, even if they do not contribute to resource utilization.
>
> Ordinarily, this command should never be used. All operations on Transfer Function coefficients can, in the usual course of things, be performed using the "Edit Memory I/O Function," <F>. The circumstances under which this command might be of value are explained below, and they depend on a more-than-superficial understanding of how PERM represents the Task Class data structures internally. Before using this command, see if the function you want to implement is not available using <F>. If it is, use <F>; if not, return here, and continue reading.

When a Segment Class is Created, PERM automatically also creates Transfer Functions -- one for every Accessible Bus variable in the associated Processor Class, and two (one for size requirements, and one for I/O requirements) for each Accessible Memory variable. These transfer functions are created so as to automatically correctly agree with the Processor Class of the Segment in both number and "type" (where "type" means variable name and Class).

In the ordinary course of things, this association between the Segment Class and the Processor Class would not change, and neither would the properties of the Processor Class. Thus, the Transfer Functions originally created should remain internally consistent: their value might change, but the variable name and Class would not. Thus, there would be no need to Delete a Transfer Function -- even a "Null" one. To do so would be to doom the Task Class data structure to fail verification.

In can happen, however, that this correspondence between the "type" of the Transfer Functions and the associated Processor Class is broken. This can happen in two ways. First, the Processor Ensemble, itself, can be edited, and the variables list and their Classes changed. If this happens, the list of Transfer Functions will no long agree, in variable name and Class, with the Accessible Bus and Memory variables of the Processor Class.

Another way that the disagreement can arise is if the Processor Class of the Segment Class is, itself, edited (using the Edit Segment Header function). In this case, the list of Accessible Bus and Memory variables of the new Processor Class need no longer agree with the variables list of the original Processor Class.

What this function is intended to do is to enable a knowledgeable user to create his own list of Transfer Functions for each of the new variables. He can combine the "Add" function with the "Delete" function to take away Transfer Functions for which variables no longer exist, and to create Transfer Functions for variables that have been introduced, and that differ from the original list of Transfer Functions created at the time the Segment Class itself was created.
In order to pass verification, the resulting list of Transfer Functions must have the following properties:

(1) each transfer function is associated with the correct name and Class of some Accessible Bus or Memory variable from the Processor Class associated with the Segment Class.

(2) for every Accessible Bus variable in the Processor Class, there exists exactly one Bus Bandwidth Transfer Function with the correct variable name and the correct Bus Class.

(3) for every Accessible Memory variable in the Processor Class, there exist exactly two Transfer Functions -- one for Memory Requirements, and one for Memory Bandwidth -- each with the correct variable name and the correct Memory Class.

Condition (1) says that every Transfer Function is associated with a variable from the Segment's Processor Class. Conditions (2) and (3) say that every variable -- whether Bus or Memory -- has exactly the right number and "type" of associated Transfer Functions.

Thus, any Transfer Function that fails to satisfy (1) - (3) can be deleted (using the "Delete" commands: <D>, <G>, and <J>), and any that are missing can be added (using the "Add" commands: <B>, <E>, and <H>).

When a "Delete" command is used, the user will be prompted to select the Task Class, Segment Class, and Transfer Function (indicated by Accessible Memory variable) to be deleted, using scrolling menus in the usual manner (<Home> and <End> to move the cursor, and <Return> to select). The execution of the command immediately causes the Transfer Function to be removed from the Task Class data structure; no "confirm" is offered or required. However, the command can be exited at any time without modification to the Task Class by entering <ESC>.

Add a Bus I/O Transfer Function (for a Task Class) (@@TEEH)

NOTE:

> Ordinarily, this command should never be used. All operations on Transfer Function coefficients can, in the usual course of things, be performed using the "Edit Bus I/O Function," <I>. The circumstances under which this command might be of value are explained below, and they depend on a more-than-superficial understanding of how PERM represents the Task Class data structures internally. Before using this command, see if the function you want to implement is not available using <I>. If it is, use <I>; if not, return here, and continue reading.

When a Segment Class is Created, PERM automatically also creates Transfer Functions -- one for every Accessible Bus variable in the associated Processor Class, and two (one for size requirements, and one for I/O requirements) for each Accessible Memory variable. These transfer functions are created so as to automatically correctly agree with the Processor Class of the Segment in both number and "type" (where "type" means variable name and Class).

In the ordinary course of things, this association between the Segment Class and the Processor Class would not change, and neither would the properties of the Processor Class. Thus, the Transfer Functions originally created should remain internally consistent: their value might change, but the variable name and Class would not.

In can happen, however, that this correspondence between the "type" of the Transfer Functions and the associated Processor Class is broken. This can happen in two ways. First, the Processor Ensemble, itself, can be edited, and the variables list and their Classes changed. If this happens, the list of Transfer Functions will no long agree, in variable name and Class, with the Accessible Bus and Memory variables of the Processor Class.

Another way that the disagreement can arise is if the Processor Class of the Segment Class is, itself, edited (using the Edit Segment Header function). In this case, the list of Accessible Bus and Memory variables of the new Processor Class need no longer agree with the variables list of the original Processor Class.

What this function is intended to do is to enable a knowledgeable user to create his own list of Transfer Functions for each of the new variables. He can combine the "Add" function with the "Delete" function to take away Transfer Functions for which variables no longer exist, and to create Transfer Functions for variables that have been introduced, and that differ from the original list of Transfer Functions created at the time the Segment Class itself was created.

In order to pass verification, the resulting list of Transfer Functions must have the following properties:

(1) each transfer function is associated with the correct name and Class of some Accessible Bus or Memory variable from the Processor Class associated with the Segment Class.

(2) for every Accessible Bus variable in the Processor Class, there exists exactly one Bus Bandwidth Transfer Function with the correct variable name and the correct Bus Class.

(3) for every Accessible Memory variable in the Processor Class, there exist exactly two Transfer Functions -- one for Memory Requirements, and one for Memory Bandwidth -- each with the correct variable name and the correct Memory Class.

Condition (1) says that every Transfer Function is associated with a variable from the Segment's Processor Class. Conditions (2) and (3) say that every variable -- whether Bus or Memory -- has exactly the right number and "type" of associated Transfer Functions.

Thus, any Transfer Function that fails to satisfy (1) - (3) can be deleted (using the "Delete" commands: <D>, <G>, and <J>), and any that are missing can be added (using the "Add" commands: <B>, <E>, and <H>).

When an "Add" command is used, the user will be prompted to select the Task Class and the Segment Class to be edited, using scrolling menus in the usual manner (<Home> and <End> to move the cursor, and <Return> to select). Then three fields will be offered for editing. The first is the name of an Accessible Resource variable -- in this case, a bus variable. The second is the Class of that variable. To pass verification, these must agree with a variable name and Class from the associated Processor Class.

NOTE:

> It is at this point that the "Add" function, <H>, differs from the "Edit" function, <I>. The "Edit" function will present another scrolling menu to select the accessible bus variable (and its Class) from the list created when the Segment Class itself was created. In the "Add" function, the user enters this information directly from the keyboard.

Third, the values of the Transfer Function coefficients must be entered. Their meaning is fully explained on the menu itself. For Bus I/O, the resulting units should be in Bytes (as a function of Input Data Size).

As with all other PERM commands, this command can be exited at any time without disturbing the Task Class data structure by entering <ESC>.

Edit a Bus I/O Transfer Function (for a Task Class) (@@TEEI)

When a Segment Class is Created, PERM automatically also creates Transfer Functions -- one for every Accessible Bus variable in the associated Processor Class, and two (one for size requirements, and one for I/O requirements) for each Accessible Memory variable. These transfer functions are created so as to automatically correctly agree with the Processor Class of the Segment in both number and "type" (where "type" means variable name and Class). Also, the Transfer Function coefficients are assigned default values, chosen so that, if left unedited, the Transfer Function effectively is null -- it will make no demands on PERM resources.

When an instance of a Segment Class is instantiated on a Thread, the Accessible Resource variables -- in this case, Accessible Bus variables -- are replaced with the "real, instantiated entities that are accessible to the "real", instantiated processor associated with the Thread. Then, as PERM runs in the computational phase, the Segments will make demands on these resources as specified by the Transfer Functions and the input data size.

Because the default values of the Transfer Functions effectively render the demands on the resource "null", the Transfer Function coefficients must be edited to reflect the actual behavior of the software block of code being modeled. The purpose of this command is to enable editing of Transfer Functions coefficients for Bus I/O -- the number of Bytes of data to be sent by the Processor on the Bus when the block of code is executed.

When the command is executed, the user will be prompted for three pieces of information: (1) the Task Class being edited; (2) the Segment Class (within the Task Class) being edited; and (3) the name of the Accessible Memory variable (from the list specified in the Processor Class definition of the Processor Class associated with the Segment Class). In each case, selection is made from a scrolling menu in the usual manner (<Home> and <End> to move the cursor, and <Return> to select). A menu is then presented with an explanation of the six coefficients that specify the Memory I/O Transfer Function, and the user edits each in turn (using the keyboard, <Arrow> keys and <Return> to move between fields and enter data). The units for output by the Transfer Function are Bytes; the number of Bytes divided by the Run Time of the Segment then yields Bandwidth, which is the quantity for which PERM does accounting. Ordinarily, Bus resource demands are specified for the sending process, not for the receiving process. The command is completed (and the Task Class data structure is updated) when the cursor located on the final field, and <Return> is entered. At any point, the user can exit the command without modification to the Task Class data structure by entering <ESC>.

Delete a Bus I/O Transfer Function (for a Task Class) (@@TEEJ)

NOTE:

> Do not use this command to get rid of Transfer Functions that are known to be "null". The default values for all Transfer Functions are set so that the Function will have no effect during PERM execution. Deleting Transfer Functions from the list will ordinarily cause the Task Class to fail verification, since it is required that all Accessible Bus and Memory variables have Transfer Functions, even if they do not contribute to resource utilization.
>
> Ordinarily, this command should never be used. All operations on Transfer Function coefficients can, in the usual course of things, be performed using the "Edit Bus I/O Function," <I>. The circumstances under which this command might be of value are explained below, and they depend on a more-than-superficial understanding of how PERM represents the Task Class data structures internally. Before using this command, see if the function you want to implement is not available using <I>. If it is, use <I>; if not, return here, and continue reading.

When a Segment Class is Created, PERM automatically also creates Transfer Functions -- one for every Accessible Bus variable in the associated Processor Class, and two (one for size requirements, and one for I/O requirements) for each Accessible Memory variable. These transfer functions are created so as to automatically correctly agree with the Processor Class of the Segment in both number and "type" (where "type" means variable name and Class).

In the ordinary course of things, this association between the Segment Class and the Processor Class would not change, and neither would the properties of the Processor Class. Thus, the Transfer Functions originally created should remain internally consistent: their value might change, but the variable name and Class would not. Thus, there would be no need to Delete a Transfer Function -- even a "Null" one. To do so would be to doom the Task Class data structure to fail verification.

It can happen, however, that this correspondence between the "type" of the Transfer Functions and the associated Processor Class is broken. This can happen in two ways. First, the Processor Ensemble, itself, can be edited, and the variables list and their Classes changed. If this happens, the list of Transfer Functions will no long agree, in variable name and Class, with the Accessible Bus and Memory variables of the Processor Class.

Another way that the disagreement can arise is if the Processor Class of the Segment Class is, itself, edited (using the Edit Segment Header function). In this case, the list of Accessible Bus and Memory variables of the new Processor Class need no longer agree with the variables list of the original Processor Class.

What this function is intended to do is to enable a knowledgeable user to create his own list of Transfer Functions for each of the new variables. He can combine the "Add" function with the "Delete" function to take away Transfer Functions for which variables no longer exist, and to create Transfer Functions for variables that have been introduced, and that differ from the original list of Transfer Functions created at the time the Segment Class itself was created.

In order to pass verification, the resulting list of Transfer Functions must have the following properties:

(1) each transfer function is associated with the correct name and Class of some Accessible Bus or Memory variable from the Processor Class associated with the Segment Class.

(2) for every Accessible Bus variable in the Processor Class, there exists exactly one Bus Bandwidth Transfer Function with the correct variable name and the correct Bus Class.

(3) for every Accessible Memory variable in the Processor Class, there exist exactly two Transfer Functions -- one for Memory Requirements, and one for Memory Bandwidth -- each with the correct variable name and the correct Memory Class.

Condition (1) says that every Transfer Function is associated with a variable from the Segment's Processor Class. Conditions (2) and (3) say that every variable -- whether Bus or Memory -- has exactly the right number and "type" of associated Transfer Functions.

Thus, any Transfer Function that fails to satisfy (1) - (3) can be deleted (using the "Delete" commands: <D>, <G>, and <J>), and any that are missing can be added (using the "Add" commands: <B>, <E>, and <H>).

When a "Delete" command is used, the user will be prompted to select the Task Class, Segment Class, and Transfer Function (indicated by Accessible Bus variable) to be deleted, using scrolling menus in the usual manner (<Home> and <End> to move the cursor, and <Return> to select). The execution of the command immediately causes the Transfer Function to be removed from the Task Class data structure; no "confirm" is offered or required. However, the command can be exited at any time without modification to the Task Class by entering <ESC>.

Edit a Thread Header (for a Task Class) (@@TEKA)

This command allows a user to change the name of a Thread. The processor, and its class, cannot be edited once the Thread has been created. Only only way to perform such an operation is to Create a new Thread (perhaps after deleting the existing one).

The user will be prompted for two pieces of information: (1) the Task Class to be edited; and (2) the Thread name within that Task Class. These selections are made from scrolling menus in the usual manner (<Home> and <End> to move the cursor, and <Return> to select). A menu is then displayed, and the Thread Name field is available for editing. Usual PERM restrictions apply: Thread names within a Task Class should all be unique, and up to 32 characters (including special characters) are permitted.

The command can be exited without altering the Task Class data structure by entering <ESC>.

Append a Segment (to a Thread, of a Task Class) (@@TEKB)

A Thread consists of a sequence of Segments. Each Segment belongs to a Segment Class (of the same Processor Class as the processor associated with the Thread). Segments in the Thread have their own names in addition to the name of the Segment Class to which they belong, and they inherit the properties (Type, Transfer Functions) of their parent Segment Class. More than one segment on a Thread (or between Threads) can belong to the same Segment Class.

This command adds a new segment to the bottom of the current list of Segments in the Thread. The user must specify three things: (1) The Task Class being edited; (2) the Thread within that Class; and (3) the Segment Class of the Segment to be added. These selections are made from scrolling menus in the usual fashion (<Home> and <End> to move the cursor, and <Return> to select). Then, a menu is presented, and the user can enter the name of the Segment. The only restriction is that Segment names within a Thread should all be unique. One approach is to include the Number (= position) of the Segment as part of its name, but that is optional. Up to 32 characters are permitted, including special characters.

If the Type of the Segment is Application Code or Operating System, the command is complete, and a segment with the designated name and Class is appended to the end of the Thread. However, if the Segment is a Join Segment, two additional pieces of information are required.

The first is an offset time (default = 0.0) to enable the analyst to enforce a minimum start time on the Join Segment's successor. The offset is always with relation to Task start time.

The second is to name .he Thread/Segment on which the Join Segment depends (that is, whose completion is required before the Join segment can complete, and its successor begin). The choice is made from scrolling menus. First, the other current Threads in the Task Class are displayed; then, when the Thread is chosen, the list of Segments (by Segment Name) for that Thread is displayed. The selection is made in the usual manner (<Home> and <End> to move the cursor, and <Return> to select). This means that only Segments that have already been defined can be selected as predecessors of a Join Segment, and this can impose an order on the sequence in which the Threads are created and edited.

Insert a Segment (into a Thread, of a Task Class) (@@TEKC)

This command enables an analyst to insert a Segment into an existing sequence of Segments constituting a Thread. To perform this operation, four pieces of information are required: (1) the Task Class to be edited; (2) the Thread within that Class; (3) the Segment Class of the Segment to be inserted; and (4) the name of the Segment prior to which the Segment is to be inserted. These choices are made using a sequence of scrolling menus in the usual manner (<Home> and <End> to move the cursor, and <Return> to select).

Next, a menu is presented, and the user can enter the name of the Segment. The only restriction is that Segment names within a Thread should all be unique. One approach is to include the Number (= position) of the Segment as part of its name, but that is optional. Up to 32 characters are permitted, including special characters.

If the Type of the Segment is Application Code or Operating System, the command is complete, and a segment with the designated name and Class is inserted at the designated position in the Thread. However, if the Segment is a Join Segment, two additional pieces of information are required.

The first is an offset time (default = 0.0) to enable the analyst to enforce a minimum start time on the Join Segments successor. The offset is always with relation to Task start time.

The second is to name the Thread/Segment on which the Join Segment depends (that is, whose completion is required before the Join segment can complete, and its successor begin). The choice is made from scrolling menus. First, the other current Threads in the Task Class are displayed; then, when the Thread is chosen, the list of Segments (by Segment Name) for that Thread is displayed. The selection is made in the usual manner (<Home> and <End> to move the cursor, and <Return> to select). This means that only Segments that have already been defined can be selected as predecessors of a Join Segment, and this can impose an order on the sequence in which the Threads are created and edited.

Edit a Segment (in a Thread, of a Task Class) (@@TEKD)

This function permits the user to change the name of a Segment in a Thread, and to alter the minimum offset time and the predecessor Thread/Segment if the Segment is a Join Segment. Three pieces of information are required: (1) the Task Class being edited; (2) the Thread within that Class; and (3) the Segment (by Segment name) within that Thread. These selections are made from scrolling menus in the usual manner (<Home> and <End> to move the cursor, and <Return> to select). A menu is then presented that permits the user to edit the Segment Name. The usual PERM restrictions apply: up to 32 characters, and Segment names within the Thread should be unique.

If the Type of the Segment is Application Code or Operating System, the command is complete. However, if the Segment is a Join Segment, two additional pieces of information can be edited.

The first is the minimum offset time (enables the analyst to enforce a minimum start time on the Join Segment's successor). The offset is always with relation to Task start time.

The second is the Thread/Segment on which the Join Segment depends (that is, whose completion is required before the Join segment can complete, and its successor begin). The choice is made from scrolling menus. First, the other current Threads in the Task Class are displayed; then, when the Thread is chosen, the list of Segments (by Segment Name) for that Thread is displayed. The selection is made in the usual manner (<Home> and <End> to move the cursor, and <Return> to select). This means that only Segments that have already been defined can be selected as predecessors of a Join Segment, and this can impose an order on the sequence in which the Threads are created and edited.

Fields within the menu can be visited directly using the <Arrow> keys. Further, the command can be exited at any time without change to the Task Class data structure by entering <ESC>.

An important use of this Command is when a sequence of Segments is copied from one Thread to another. Often, the analyst will want to rename some of the Segments, and will also want to modify the dependency structure of the Join Segments (if any).

Delete a Segment (from a Thread, of a Task Class) (@@TEKE)

This command allows a Segment to be deleted from the list of Segments constituting a Thread. Three pieces of information are required: (1) the Task Class to be edited; (2) the Thread within the Task Class; and (3) the name of the Segment to be deleted. The user is prompted for this information on a sequence of scrolling menus, and the selection is made in the usual manner (<Home> and <End> to move the cursor, and <Return> to select). As soon as the final <Return> is entered, the selected Segment is immediately deleted from the list. No "confirm" is offered or required.

It is perfectly permissible to delete a Segment from the middle of the list of Segments.

Display the Segment List (for a Thread, of a Task Class) (@@TEKF)

Use of this command scrolls to the screen the list of Segments belonging to the selected Thread. The information includes:

- Thread Name
- Processor Name (instantiated) and Class
- Segment List
  - For each Segment
    - Segment Name
    - Segment Class
    - Segment Type (Join, OS, AC)
    - For Join Segments,
      - Predecessor Thread and Segment

Two pieces of information are required: (1) the Task Class, and (2) the Thread within the Class. The user is prompted for the selection from scrolling menus, and the selection is made in the usual manner (<Home> and <End> keys to move the cursor, and <Return> to select). The data is presented on the screen by pages, with <c> available to stop the display at any time.

2.3.1.2.3 Help Files for System Load Operations

Load Operations (@@L)

This top-level command opens onto a set of sub-menus that permit the user to assemble and verify a System Load suitable for input to the PERM Compute component.

NOTE:

> The Load menu requires that a validated Processor Ensemble and zero or more validated Task Classes be in memory. The validation date of the Task Classes should be later than the validation date of the Processor Ensemble. If these data structures are not already in memory when Load is executed, the Load operations will not function. To get them into memory, Processor Ensemble operations (Load, or Create) and Task Class operations (Load, or Create) must be exercised.

The PERM commands are primarily for data entry. The analyst should already have created coherent Processor Ensemble and Task Class models, including mnemonic naming conventions, before interacting with the PERM user interface.

Create a System Load (@@LC)

PERM only supports one System Load in memory at a time. There are two ways in which a System Load can be placed into memory: (1) via the Create command; or (2) using the Load command to load the data structure into memory from disk. Neither command will work if a System Load data structure already exists in memory. That data structure must first be removed (using the Remove command); then Create or Load can be used.

The Create command does nothing but create a Header for the Load. A menu is presented with three fields: (1) the name of the System Load; (2) the author; and (3) the maximum (modeling) run time the Load is permitted to achieve. The first two fields are PERM standard: 32 characters including special characters, with <Arrow> keys to move between fields, and <Return> to enter.

The third field is entered as floating point, and expresses the time (in modeling seconds) at which the PERM compute phase should stop processing. That is, during Compute, a "current time" pointer is kept to indicate how far through the Tasks and Threads the processing has proceeded. PERM will use the "maximum time" value as a ceiling beyond which this pointer cannot proceed. The units of this parameter are Seconds.

As with other PERM commands, this one can be exited without having any effect by using the <ESC> key at any point. The command itself executes when the <Return> key is entered while the cursor is in the bottom field.

Edit a System Load (@@LE)

This command opens onto a submenu that provides functions to instantiate Tasks (from Task Classes) and define their data size values and inter-task dependencies. A System Load data structure must be in memory when this command is executed -- either from use of the Create command, or from use of the Load command.

Remove a System Load from Memory (@@TR)

This command clears memory of the current System Load data structure. Since only one such data structure can be in memory at a time, it requires no further input from the user; no "confirm" is required or accepted.

NOTE:

> Care should be used with this command, since it results in the loss of the data structure. If there is any possibility that the data may be wanted for later use, it should first be saved to permanent storage using the Save command.

Display a System Load Data Structure (@@LD)

This command prints to the terminal an abbreviated ASCII account of the current System Load data structure. This includes Header information followed by a list of all instantiated Tasks and their forward (Output) and backward (Input) dependencies. A complete listing, including the Processor Ensemble description and a Task Class description for all Tasks is available through the Print command. The <C> key can be used at any time to stop the listing and exit the command.

Load a System Load Data Structure from Disk (@@LL)

This command works together with the Save command to allow the user to Save and Restore data structures to and from permanent storage on disk. The user names the file from which the data is to be loaded. A full DOS path name, including drive and directory, can be used. If omitted, PERM will use the default directory (the directory from which PERM was executed).

The file must have been created using the Save command. PERM checks to ensure that the contents of the file do, indeed, constitute a Load data structure. Otherwise, it writes an error message, and awaits further action by the user. It responds similarly if it cannot find or open the named file.

Save a System Load Data Structure to Disk (@@LS)

This command works together with the Load command to enable the user to Save and Restore a Load data structure to and from permanent storage. It can be used, for example, when an editing session must be interrupted; the saved data can then later be restored (using Load), and work can continue. Similarly, when a complete PERM model is achieved, it can (and should) be saved so that it can be revisited and modified as appropriate during an engineering study.

If the System Load is complete and verified, the file created by this command can be used as input to CPERM.

The user will be prompted for the name of the file to use to store the data. A full DOS path name can be used, including drive and directory path. If this is omitted, PERM will use the default directory (the directory the user was in when PERM was executed). If PERM cannot open the file, an error message will be returned. Also, note that if a file of the same name already exists, it will be over-written (and hence, lost) when this command is executed.

NOTE:

> There is an important difference between the Save command and the Print command. The Save command writes a binary copy of the PERM data structure to the file -- a copy which can later be loaded back into memory, and on which PERM operations will work. The Print command, on the other hand, writes a human-readable ASCII file of the data structure. This can then be reviewed and edited, but it cannot be loaded back into memory for operation by PERM. If you wish to save your work for later use -- use Save. If you wish to look at your work in ASCII form, use Print.

Print the System Load Data Structure (@@LP)

This command prints to a user-selected file an ASCII version of the full System Load data structure. The file can then be operated on by DOS utilities and programs -- for example, to route the file to a printer, or to edit or review it using a word processor. The contents include:

* System Load Header information;
* a complete listing for the Processor Ensemble, including Processor, Memory and Bus Class definitions and instantiations; and

* for each instantiated Task, a complete listing for the associated Task Class, including Segment Classes and Threads.

The user will be prompted for the name of the file to use to store the data. A full DOS path name can be used, including drive and directory path. If this is omitted, PERM will use the default directory (the directory the user was in when PERM was executed). If PERM cannot open the file, an error message will be returned. Also, note that if a file of the same name already exists, it will be over-written (and hence, lost) when this command is executed.

NOTE:

> There is an important difference between the Save command and the Print command. The Save command writes a binary copy of the PERM data structure to the file -- a copy which can later be loaded back into memory, and on which PERM operations will work. The Print command, on the other hand, writes a human-readable ASCII file of the data structure. This can then be reviewed and edited, but it cannot be loaded back into memory for operation by PERM. If you wish to save your work for later use -- use Save. If you wish to look at your work in ASCII form, use Print.

Verify a System Load Data Structure (@@LV)

This command validates the internal consistency of a complete System Load. Inconsistencies generate error messages which identify for the user the type and location of the error. Only a verified System Load can be input to the PERM Compute component. Since both Processor Ensemble and Task Class data structures have previously been verified, any errors should be due to mis-matches between the Task Class of a dependency variable and the Class of the instantiated Task that is assigned to the variable.

Edit the Header of a System Load (@@LEA)

This command allows the user to review or edit the header information that was entered when the System Load was first Created (or last edited). Fields for editing include the name of the System Load, the author, and the maximum model time (entered as floating point, in Seconds; exceeding this model time causes the PERM Compute phase to exit). As usual, the <Arrow> keys move the cursor between fields and to characters within a field for over-typing. The <Return> key records the data, and a <Return> on the final field causes the command to execute, recording the changes in the System Load data structure. The <ESC> key can be used to exit the command at any point without altering the System Load data structure.

Create a Random Variable (@@LEB)

[This capability, which supports a Monte Carlo version of PERM, is not currently implemented.]

Edit a Random Variable (@@LEC)

[This capability, which supports a Monte Carlo version of PERM, is not currently implemented.]

Delete a Random Variable (@@LED)

[This capability, which supports a Monte Carlo version of PERM, is not currently implemented.]

Instantiate a Task (@@LEE)

This command creates a instance of a Task Class. The user selects the Task Class to be instantiated, and then assigns a name to the instance. The selection of the Task Class is from a scrolling list of the Task Classes currently in memory (<Home> and <End> move the cursor, and <Return> selects). A menu is then presented to allow the user to enter the Task name. The names of all Tasks within a System Load should be unique; failure to adhere to this rule will cause the System Load to fail verification. The name may be up to 32 characters, including special characters.

Once the Task has been created, it can be edited to specify the values of the predecessor and successor Task dependencies.

Edit the Header of an Instantiated Task (@@LEF)

The only field for editing in an instantiated Task is its name. The user selects the Task to the edited from a scrolling menu of the currently instantiated Tasks (<Home> and <End> move the cursor, and <Return> selects). A menu is then presented for editing. The command can be exited without change to the System Load data structure by entering <ESC>. Thus, the command can be used to review the Header information (including, for example, the Task Class of the instantiated Task).

Edit an Input Dependency (@@LEG)

A System Load consists of a directed graph of instantiated Tasks. Heuristically, in the Compute phase of PERM, this means that a Task cannot begin execution until all its immediate predecessors (its Input Dependencies) have been satisfied. This connectivity -- forward and backward -- between nodes must be entered by the analyst. This is accomplished in the following manner.

Each Task belongs to a Task Class, and inherits the properties of that Task Class. In particular, each Task Class has been assigned Input and Output Dependency variables. These act as "place-holders", and each one must be replaced by the name of an instantiated Task during System Load definition. To express the same idea in another way, the "generic" forward and backward connectivity provided by the variables is "instantiated" by assigning to each connectivity variable the name of an instantiated Task.

NOTE:

> It will be apparent that there is redundancy in specifying both the forward (Output) and backward (Input) Task connectivity patterns. This redundancy is an artifact of PERM's current prototype status. PERM does not verify whether or not the connectivity pattern specifies a directed graph without cycles. Failure to adhere to this rule can result in deadlock or looping during the PERM compute phase.

The Input and Output Dependency variables that were created for the Task Class are, themselves, associated with Task Classes. That is, only Tasks belonging to the same Task Class as the variable can permissibly be assigned to that variable. The System Load Verify function checks this property.

The purpose of this command, then, is to enable the user to establish the backward (that is, Input) dependencies for an instantiated Task. This is done variable-by-variable; that is, for each input variable in the Task Class of the instantiated Task being edited, the user specifies the name of an instantiated (or, to be instantiated) Task.

When the command is executed, the user must first specify two things: (1) the name of the instantiated Task whose input connectivity is to be edited; and (2) the name of the Input Dependency variable (belonging to the Task Class of that Task) whose values is to be specified. These are selected from scrolling menus in the usual manner (<Home> and <End> keys to move the cursor, and <Return> to select). A menu containing one editable field -- the name of the Task to be assigned to the variable -- is then presented. The name should agree exactly with the name of an instantiated (or, to be instantiated) Task. During verify, PERM will check to ensure that the Class of the named Task agrees with the Class of the variable to which it was assigned.

NOTE:

> To provide some flexibility, PERM provides a "wild card" capability. The character string "dummy" (all lower-case) matches all Task Classes, and effectively tells the Compute phase of PERM that, in this instantiation of the Task, no Input dependency is required. For example, this enables a Task that appears in the midst of the directed graph (with both predecessors and successors) to share a Task Class with a Task that heads the graph (no predecessors) or that terminates the graph (no successors).

Edit an Output Dependency (@@LEH)

A System Load consists of a directed graph of instantiated Tasks. Heuristically, in the Compute phase of PERM, this means that a Task cannot begin execution until all its immediate predecessors (its Input Dependencies) have been satisfied. This connectivity -- forward and backward -- between nodes must be entered by the analyst. This is accomplished in the following manner.

Each Task belongs to a Task Class, and inherits the properties of that Task Class. In particular, each Task Class has been assigned Input and Output Dependency variables. These act as "place-holders", and each one must be replaced by the name of an instantiated Task during System Load definition. To express the same idea in another way, the "generic" forward and backward connectivity provided by the variables is "instantiated" by assigning to each connectivity variable the name of an instantiated Task.

NOTE:

> It will be apparent that there is redundancy in specifying both the forward (Output) and backward (Input) Task connectivity patterns. This redundancy is an artifact of PERM's current prototype status. PERM does not verify whether or not the connectivity pattern specifies a directed graph without cycles. Failure to adhere to this rule can result in deadlock or looping during the PERM compute phase.

The Input and Output Dependency variables that were created for the Task Class are, themselves, associated with Task Classes. That is, only Tasks belonging to the same Task Class as the variable can permissibly be assigned to that variable. The System Load Verify function checks this property.

The purpose of this command, then, is to enable the user to establish the forward (that is, Output) dependencies for an instantiated Task. This is done variable-by-variable; that is, for each Output variable in the Task Class of the instantiated Task being edited, the user specifies the name of an instantiated (or, to be instantiated) Task.

When the command is executed, the user must first specify two things: (1) the name of the instantiated Task whose input connectivity is to be edited; and (2) the name of the Output Dependency variable (belonging to the Task Class of that Task) whose values is to be specified. These are selected from scrolling menus in the usual manner (<Home> and <End> keys to move the cursor, and <Return> to select). A menu containing one editable field -- the name of the Task to be assigned to the variable -- is then presented. The name should agree exactly with the name of an instantiated (or, to be instantiated) Task. During verify, PERM will check to ensure that the Class of the named Task agrees with the Class of the variable to which it was assigned.

NOTE:

> To provide some flexibility, PERM provides a "wild card" capability. The character string "dummy" (all lower-case) matches all Task Classes, and effectively tells the Compute phase of PERM that, in this instantiation of the Task, no Output dependency is required. For example, this enables a Task that appears in the midst of the directed graph (with both predecessors and successors) to share a Task Class with a Task that heads the graph (no predecessors) or that terminates the graph (no successors).

Delete an Instantiated Task (@@LEI)

This command deletes an instantiated Task from the System Load data structure. The user is prompted to select the Task to be deleted from a scrolling menu of the currently instantiated Tasks (<Home> and <End> to move the cursor, and<Return> to select). No "confirm" is required or accepted, but the user can exit the command without affecting the System Load data structure by entering <ESC>.

Clear (the Instantiated Tasks from) a System Load (@@LEJ)

This command immediately deletes all currently instantiated Tasks from the System Load data structure. Its effect is as if the Delete Task command had been applied separately to each instantiated Task. The resulting state is like that of a Load immediately after it is created, but before any Tasks have been instantiated for it. The command takes effect immediately, and no "confirm" is required or accepted. Thus, the same cautions that apply to the Remove command (on the main Load menu) apply here: if there is any possibility that the data currently in the System Load data structure will be needed later, it should first be Saved to disk (using the Save command from the top level menu).

Load data structure will be needed later, it should first be Saved to disk (using the Save command from the top level menu).

2.3.1.2.4 Miscellaneous Commands

Open a Log File (@O)

This command enables the user to Log (that is, print) all screen activity, in ASCII form, to a file of his choosing. That is, from the point at which the Open Log File command is issued until the Close Log File command is issued (or PERM is exited), a record will be made in the file of any text that PERM writes to the screen. Key strokes and menu manipulations are not recorded -- only messages that scroll to the screen.

The kinds of information that might be useful include:

* error messages from the Verify functions; and
* copies of the lower-level Display commands (e.g., Display thread, Display Instantiations).

The latter are useful, since the PERM Print function gives a complete listing of the entire data structure (Processor Ensemble, Task Class, or System Load). If only a portion of the structure -- that offered by a lower-level Display command -- is wanted, then the Log File can be used for that purpose.

The user will be prompted for the name of the DOS file to be opened, and to which the messages will be printed. A full DOS path name, including drive and directory, can be used. If this is omitted, PERM will use the default directory (the directory from which PERM was executed). If PERM cannot open the file, an error message is printed. If the file already exists, PERM will over-write it, and its contents will be lost. Logging will continue until either the Close Log File command is issued, or until PERM is exited.

Close the Log File (@C)

If the user has opened a Log file using the Open Log File command, this command can be used to close it. All logging to that file ceases. No user input is required.

Exit PERM (@Q)

This command causes PERM to exit, and return to the DOS prompt in the directory from which PERM was executed.

NOTE:

All data currently in memory is lost. If there is any possibility that these data structures -- Processor Ensemble, Task Class, or System Load -- will be needed at some future time, they should be saved to permanent storage using the Save commands in the <P>, <T>, of <L> sub-menus.

No "confirm" is required or accepted. Once <Return> is entered, PERM is immediately exited, and all data structures are lost.

2.3.2 CPERM

2.3.2.1 Running CPERM – In order to execute CPERM, do the following:

1. Place yourself in a directory. This will be your default directory. Make sure that both the files cperm.hlp and cperm.ndx are in this directory. Also, if the file cperm.exe or the file created during IPERM of a verified System Load is not in this directory, make sure you know where they are, including their full DOS path name.

2. Enter cperm.exe, including its full DOS path if it is not in your current, default directory.

3. When you exit CPERM, you will find that three new files have been created, with extensions: .INX, .PE, and .EHF . These files will be needed by DPERM if you wish to display the data.

2.3.2.2 Help Files for CPERM

Load a System Load Data Structure into Memory (@@L)

Before any other Compute operations can take effect, a System Load data structure (as created by the Save command under the Load Operations menu in the PERM initialization phase) must be loaded into memory. The user enters the name of that file. A full DOS path name can be used, including drive and directory; otherwise, PERM will use the default directory -- the directory from which PERM was executed. If PERM cannot open the file, or if the file does not contain a recognizable System Load data structure, an error message is displayed, and PERM waits for further user action.

Verify the System Load Data Structure (@@V)

Once a System Load data structure has been loaded into memory using the Load command, it must be verified for internal consistency. This is done even if the structure was previously verified in the Load Operations menu of the initialization phase of PERM. Any errors detected by the verification function are printed to the screen. The <C> key can be used to exit the display and return to the menu. If errors are found, they must first be corrected by exiting the compute phase of PERM, and returning to the initialization phase of PERM. Only a verified System Load data structure can be used by the PERM compute operation.

Compute (@@C)

This command causes the resource utilization statistics associated with the System Load data structure to be computed and stored to permanent files for display by the Display phase of PERM. The user will be required to identify the output files, and will need to remember this identification information so that it can be re-entered when PERM display routines are run.

The file identification information for this command is different from that for many other PERM commands. In particular, the user cannot specify a full DOS path name, and cannot use any extension. The reason is that PERM will actually create three files for use in Display, and these files must (1) reside in the default directory, and (2) have the proper extensions.

The user will be prompted for a file identifier -- up to eight ASCII alphanumeric characters acceptable to DOS as a file name. The compute command will then use this identifier as the prefix for three different files with extensions .INX, .PE, and .EHF, respectively.

EXAMPLE:

The user executes CPERM from the DOS file

c:\perm_run

When executing the <C> command, in response to the prompt, he enters the file name prefix

TRACKING

When the Compute command is complete, three files will have been created in the c:\perm_run directory:

TRACKING.INX
TRACKING.PE
TRACKING.EHF

These three files must reside in the default directory when the Display phase of PERM (DPERM) is executed, and the user will again be prompted for the (up to) eight character prefix (in this example, TRACKING).

Administrative Commands (@@A)

This command opens onto a submenu that controls the log file and the scrolling of CPERM operational status to the screen.

Open Log File (@@AA)

This command opens a user-selected DOS file to which is written all messages to the screen. Thus, for example, if the user chooses to have the Event Display set to "ON", then not only will the events be written to the screen, they will also be written to the log file. In this way, the event history can be examined at leisure, in hard-copy.

The user will be prompted for a DOS file name. A full DOS path can be used, including drive and directories; if this is omitted, the default directory will be used. If the named file cannot be opened, the user will be notified with an error message, and CPERM will await further instruction. If the names file already exists, it will be over-written, and its contents lost.

Close Log File (@@AB)

This command closes the log file opened by the Open Log File command. No additional user input is required. Once the command is issued, screen output is no longer routed to the file. Also, the file is automatically closed when CPERM is exited, whether or not the Close File command has been issued.

Event Display On (@@AC)

This command is one of a pair of switches. In the "ON" condition, which is the default, the model time of each event in the Event History File is scrolled to the screen (if the "Paged" switch is on, the scrolling is done a screenful at a time; otherwise, the data scrolls continuously). Each record also contains the name of the segment that generated the event, and its processor.

NOTE:

> The default is to "ON". Thus, no action is necessary if the user wants the events scrolled to the screen.

Event Display Off (@@AD)

This switch causes no event data to be scrolled to the screen. Since the default is to "ON", this switch must be exercised if no event data output is desired.

Transition Display On (@@AE)

This switch causes the state transitions of the Finite State Automata that drives CPERM to be printed to the screen, in the same manner that the events are printed when the Event Display switch is "ON". If both switches are "ON", then both sets of data are interspersed and sent to the screen in a time-ordered sequence. This data is primarily intended for debug purposes, and should not be of interest to most users.

NOTE:

> The default for this switch is "OFF". Thus, the user must exercise this command in order to have the data for review.

Transition Display Off (@@AF)

This switch disables the scrolling to the screen of the Finite State Automata transition data. Since the default for this switch is already "OFF", it is only necessary to exercise this command if its companion, "ON", switch has previously been exercised.

Display Paging On (@@AG)

There are two ways in which the data from the previous two switches (Events and Transitions) can be scrolled to the screen: a page at a time, (with the standard <c> key to cancel output); and, continuously. The paged option, which is set by using this command, enables the user to examine the current screenful before proceeding to the next. The other option is to have the data continuously scrolled to the screen. Continuous scrolling is the default, so it is not necessary to use this command unless the user wants to examine the data a page at a time.

Display Paging Off (@@AH)

There are two ways in which the data from the previous two switches (Events and Transitions) can be scrolled to the screen: a page at a time, (with the standard <c> key to cancel output); and, continuously. This command selects the "continuously scrolled" option, and is the default. The data will move past too quickly for review. However, if the "Open Log" command has been previously exercised, the data will also be routed to the log file for more leisurely perusal at a later time.

NOTE:

> Paging "OFF" is the system default. Thus, if the user wants continuous scrolling, he need take no action. This command is only needed if the "Paging" option has been previously selected, and now it is desired to return to the default.

Remove the System Load Data Structure (@R)

This command clears memory of the currently resident System Load data structure so that another one can be loaded in. No user input is required.

Exit the PERM Compute Phase (@Q)

This command causes PERM to exit, and returns the user to the DOS prompt from which PERM was executed. No "confirm" is offered or accepted.

2.3.3 DPERM

2.3.3.1 Running DPERM – In order to execute DPERM, do the following things:

1. Place yourself in a DOS directory. This will be your default directory. Make sure that both the file dperm.hlp and dperm.ndx are in this directory. Also, make sure that there are three files with the same name but different extensions: .INX, .PE, and .EHF. These should have been created by CPERM, and (if necessary) moved or copied into your default directory. If the file dperm.exe is not in this directory, make sure you know where it is, including its full DOS path name.

2. Enter dperm.exe, including its full DOS path name if it is not already in your current directory.

2.3.3.2 Help Files for DPERM

Set Print Driver (@P)

This command informs PERM of the type of Printer attached to your PC/AT. PERM assumes this printer is attached to the output port LPT1. It uses this output port to drive all Graphics hardcopy outputs, and the purpose of this command is to match the Printer Driver to the type of Printer being used.

The user selects from the available drivers in the usual manner, using <Home> and <End> to move the cursor, and <Return> to select.

Load Display Data From Disk (@L)

The Display phase of PERM requires files that have been produced by the Compute phase, and a file naming convention has been built into the software to facilitate this hand-off. In particular, the Display component of PERM *requires that three files with specified DOS extension* be resident in the default directory when DPERM is executed. The file names for these three files all agree, and are entered by the user. The extensions, however, must be exactly: .INX, .PE, and .EHF. The Display software itself adds these extensions, and searches for and Reads the resulting files. During the Compute phase of PERM, the user was similarly prompted for and extension less DOS file name, and it was this name that PERM used to create the three files (in the default directory). Now, these files are being read back into memory for use by the Display component.

EXAMPLE:

The user executes CPERM from the DOS file

c:\perm_run

When executing the <C> command, in response to the prompt, he enters the file name prefix

TRACKING

When the Compute command is complete, three files will have been created in the c:\perm_run directory:

TRACKING.INX
TRACKING.PE
TRACKING.EHF

These three files must reside in the default directory when the Display phase of PERM (DPERM) is executed. When the Load <L> command is executed, the user is prompted for the (up to) eight character prefix (in this example, TRACKING). It is up to the user to remember the eight character extension less file name originally entered during the Compute phase of PERM. The three files, with the correct extensions, are then Read back into memory for use by Display.

Remove Display Data From Memory (@R)

This command clears memory of the current PERM model being examined so that another can be loaded, if desired. No user input is required.

Display PERM Resource Utilization Data (@D)

This command opens onto a sequence of sub-menus that provide the capability to examine the performance and resource utilization data generated by the PERM modeling capability. Four types of data are provided:

* a system-wide history of all segment activity on all processors;
* the subset of the history file that relates to a single processor;
* the resource utilization profile for any bus or memory; and
* for any bus or memory, its utilization by any selected processor.

For all four types, the user can specify a sub-interval of time over which data is to be displayed. And, in addition, for the last two types of displays, the user can generate graphical output, both to the screen and to the printer.

Detailed explanations of these capabilities are provided with the commands that perform them.

Exit the PERM Display Phase (@Q)

The command terminates PERM software activity, and returns the user to the DOS prompt. No further user input is required; in particular, no "confirm" is offered or accepted.

Display Event List (@@DE)

This command opens onto a sub-menu which allows the user to select whether output should be directed to the screen or to a DOS disk file. Also, the user will be able to choose the time interval over which the data should be displayed.

During the Compute phase of PERM, a record is made of the following events during the course of the simulation:

* the start and end of each segment, including segment run time and resource utilization as derived from the Transfer Functions; and

* the start and end of each Task.

The purpose of this command is to display (or print) this file in ASCII form for review by the user. Since all other PERM displays are derived from this file, all data about system behavior is contained here (perhaps implicitly). Thus, if the user wants to verify the behavior of the system at or around a critical time value, for example, he has access to the full "raw" data set produced by PERM.

The data output by this command consists of the following:

1. A Header that names
   - the Compute file from which the data was drawn
   - the Processor Ensemble name
   - the Author
   - the System Load Epoch (maximum end time)
   - the start and end times selected by the user when the command was entered

2. For each Segment Boundary Event (the start or end of a segment):
   - the Type of the segment following the event (Application Code, OS, Join)
   - the time of the event
   - the duration of the segment following the event
   - the name and Class of that segment

- the Task and Task Class of that segment
- the Thread to which that segment belongs
- for each memory resource
  -- the name and Class of the memory
  -- the amount of memory used, as a percent of total capacity, both preceding and following the boundary event
  -- the memory bandwidth, as a percent of total bandwidth, both preceding and following the boundary event

NOTE:

Memory bandwidth is obtained by dividing the Run Time of the segment (in Seconds, derived from the Run Time Transfer Function) into the Memory I/O of the segment (in Bytes, derived from the Memory I/O Transfer Function). Thus, PERM assumes that the memory references are evenly spread across the run time of the segment.

- for each bus resource
  -- the name and Class of the bus
  -- the amount of bus bandwidth utilized, as a percent of total bandwidth available, both preceding and following the boundary event.

NOTE:

Bus bandwidth is obtained by dividing the Run Time of the segment (in Seconds, derived from the Run Time Transfer Function) into the Bus I/O of the segment (in Bytes, derived from the Bus I/O Transfer Function). Thus, PERM assumes that the bus utilization is evenly spread across the run time of the segment.

3. For each Task Start or End
   - the name and Class of the Task
   - the time at which the event occurs

Note that, for segment boundary events, resource utilization data is reported for both sides of the event -- that is, for both the segment that has just ended and for the segment that is just beginning.

Print the Event List to a File (@@DEF)

During the Compute phase of PERM, a record is made of the following events during the course of the simulation:

* the start and end of each segment, including segment run time and resource utilization as derived from the Transfer Functions; and

* the start and end of each Task.

The purpose of this command is to print this file in ASCII form to a DOS file for review by the user. Since all other PERM displays are derived from this file, all data about system behavior is contained here (perhaps implicitly). Thus, if the user wants to verify the behavior of the system at or around a critical time value, for example, he has access to the full "raw" data set produced by PERM.

The data output by this command consists of the following:

1. A Header that names
   - the Compute file from which the data was drawn
   - the Processor Ensemble name
   - the Author
   - the System Load Epoch (maximum end time)
   - the start and end times selected by the user when the command was entered

2. For each Segment Boundary Event (the start or end of a segment):
   - the Type of the segment following the event (Application Code, OS, Join)
   - the time of the event
   - the duration of the segment following the event
   - the name and Class of that segment
   - the Task and Task Class of that segment
   - the Thread to which that segment belongs
   - for each memory resource
     - -- the name and Class of the memory
     - -- the amount of memory used, as a percent of total capacity, both preceding and following the boundary event
     - -- the memory bandwidth, as a percent of total bandwidth, both preceding and following the boundary event

NOTE:

Memory bandwidth is obtained by dividing the Run Time of the segment (in Seconds, derived from the Run Time Transfer Function) into the Memory I/O of the segment (in Bytes, derived from the Memory I/O Transfer Function). Thus, PERM assumes that the memory references are evenly spread across the run time of the segment.

   - for each bus resource
     - -- the name and Class of the bus
     - -- the amount of bus bandwidth utilized, as a percent of total bandwidth available, both preceding and following the boundary event.

NOTE:

Bus bandwidth is obtained by dividing the Run Time of the segment (in Seconds, derived from the Run Time Transfer Function) into the Bus I/O of the segment (in Bytes, derived from the Bus I/O Transfer Function). Thus, PERM assumes that the bus utilization is evenly spread across the run time of the segment.

3. For each Task Start or End
   - the name and Class of the Task
   - the time at which the event occurs

Note that, for segment boundary events, resource utilization data is reported for both sides of the event -- that is, for both the segment that has just ended and for the segment that is just beginning.

When the command is executed, the user will be prompted for the name of the file to which the data is to be written. A full DOS path name, including drive and directory, can be used. If omitted, PERM will use the default directory -- the directory from which PERM was executed. If PERM cannot open the file, and error message is displayed, and PERM waits for further action by the user. If the named file already exists, it will be over-written, and its contents lost.

Once the file name has been entered, the interval start and end times must be entered. These are entered as floating point numbers into a menu provided for this purpose. The default start time is 0.; the default end time is the Epoch length (as specified during System Load definition in the initialization phase of PERM). Both the new start and end times must lie in this interval, and the new end time must be greater than the new start time. Failure to adhere to these rules will generate an error message, and the user can either try again or exit the command, using <ESC>.

Print the Event List to the Screen (@@DES)

During the Compute phase of PERM, a record is made of the following events during the course of the simulation:

* the start and end of each segment, including segment run time and resource utilization as derived from the Transfer Functions; and

* the start and end of each Task.

The purpose of this command is to print this file in ASCII form to the screen for review by the user. As usual, the data is scrolled a page at a time, and the <C> key can be used to stop the data output and return to the menu. Since all other PERM displays are derived from this file, all data about system behavior is contained here (perhaps implicitly). Thus, if the user wants to verify the behavior of the system at or around a critical time value, for example, he has access to the full "raw" data set produced by PERM.

The data output by this command consists of the following:

1. A Header that names
   - the Compute file from which the data was drawn
   - the Processor Ensemble name
   - the Author
   - the System Load Epoch (maximum end time)
   - the start and end times selected by the user when the command was entered

2. For each Segment Boundary Event (the start or end of a segment):
   - the Type of the segment following the event (Application Code, OS, Join)
   - the time of the event
   - the duration of the segment following the event
   - the name and Class of that segment
   - the Task and Task Class of that segment
   - the Thread to which that segment belongs
   - for each memory resource
     -- the name and Class of the memory
     -- the amount of memory used, as a percent of total capacity, both preceding and following the boundary event
     -- the memory bandwidth, as a percent of total bandwidth, both preceding and following the boundary event

NOTE:

Memory bandwidth is obtained by dividing the Run Time of the segment (in Seconds, derived from the Run Time Transfer Function) into the Memory I/O of the segment (in Bytes, derived from the Memory I/O Transfer Function). Thus, PERM assumes that the memory references are evenly spread across the run time of the segment.

   - for each bus resource
     -- the name and Class of the bus
     -- the amount of bus bandwidth utilized, as a percent of total bandwidth available, both preceding and following the boundary event.

NOTE:

Bus bandwidth is obtained by dividing the Run Time of the segment (in Seconds, derived from the Run Time Transfer Function) into the Bus I/O of the segment (in Bytes, derived from the Bus I/O Transfer Function). Thus, PERM assumes that the bus utilization is evenly spread across the run time of the segment.

3. For each Task Start or End
   - the name and Class of the Task
   - the time at which the event occurs

Note that, for segment boundary events, resource utilization data is reported for both sides of the event -- that is, for both the segment that has just ended and for the segment that is just beginning.

The user will be prompted for the interval start and end times. These are entered as floating point numbers into a menu provided for this purpose. The default start time is 0.; the default end time is the Epoch length (as specified during System Load definition in the initialization phase of PERM). Both the new start and end times must lie in this interval, and the new end time must be greater than the new start time. Failure to adhere to these rules will generate an error message, and the user can either try again or exit the command, using <ESC>.

Display the Event List for a Processor (@@DD)

This command opens onto a sub-menu which allows the user to select whether output should be directed to the screen or to a DOS disk file. Also, the user will be able to choose the time interval over which the data should be displayed.

During the Compute phase of PERM, a record is made of the following events during the course of the simulation:

* the start and end of each segment, including segment run time and resource utilization as derived from the Transfer Functions; and

* the start and end of each Task.

The purpose of this command is to display (or print) the segments events relating to a particular processor, as well as summary information on processor utilization by segment Type (Application Code, Operating System, or waiting to synchronize in a Join).

The data output by this command consists of the following:

1. A Header that names
   - the name of the Processor being examined
   - the Compute file from which the data was drawn
   - the Processor Ensemble name
   - the Author
   - the System Load Epoch (maximum end time)
   - the start and end times selected by the user when the command was entered

2. For each Segment Boundary Event (the start or end of a segment):
   - the Type of the segment following the event (Application Code, OS, Join)
   - the time of the event
   - the duration of the segment following the event
   - the name and Class of that segment
   - the Task and Task Class of that segment
   - the Thread to which that segment belongs
   - for each memory resource
     - -- the name and Class of the memory
     - -- the amount of memory used, as a percent of total capacity, both preceding and following the boundary event
     - -- the memory bandwidth, as a percent of total bandwidth, both preceding and following the boundary event

NOTE:

Memory bandwidth is obtained by dividing the Run Time of the segment (in Seconds, derived from the Run Time Transfer Function) into the Memory I/O of the segment (in Bytes, derived from the Memory I/O Transfer Function). Thus, PERM assumes that the memory references are evenly spread across the run time of the segment.

   - for each bus resource
     - -- the name and Class of the bus
     - -- the amount of bus bandwidth utilized, as a percent of the total bandwidth available, both preceding and following the boundary event.

NOTE:

Bus bandwidth is obtained by dividing the Run Time of the segment (in Seconds, derived from the Run Time Transfer Function) into the Bus I/O of the segment (in Bytes, derived from the Bus I/O Transfer Function). Thus, PERM assumes that the bus utilization is evenly spread across the run time of the segment.

3. A summary of Processor Activity during the selected time interval, including
   - the name of the Processor
   - the time interval selected by the user
   - the Compute file from which the data was drawn
   - the total time, and percent of time, spent in
     -- Application Code segments
     -- Operating System segments
     -- Join segments

4. A summary of Processor Activity during the entire Epoch interval, including
   - the name of the Processor
   - the time interval selected by the user
   - the Compute file from which the data was drawn
   - the total time, and percent of time, spent in
     -- Application Code segments
     -- Operating System segments
     -- Join segments

Print Event List for a Processor to a File (@@DDF)

During the Compute phase of PERM, a record is made of the following events during the course of the simulation:

* the start and end of each segment, including segment run time and resource utilization as derived from the Transfer Functions; and

* the start and end of each Task.

The purpose of this command is to print an ASCII version of the segment boundary events relating to a particular processor, as well as summary information on processor utilization by segment Type (Application Code, Operating System, or waiting to synchronize in a Join), to a DOS file for review by the user.

The data output by this command consists of the following:

1. A Header that names
   - the name of the Processor being examined
   - the Compute file from which the data was drawn
   - the Processor Ensemble name
   - the Author
   - the System Load Epoch (maximum end time)
   - the start and end times selected by the user when the command was entered

2. For each Segment Boundary Event (the start or end of a segment):
   - the Type of the segment following the event (Application Code, OS, Join)
   - the time of the event
   - the duration of the segment following the event
   - the name and Class of that segment
   - the Task and Task Class of that segment
   - the Thread to which that segment belongs
   - for each memory resource
     -- the name and Class of the memory
     -- the amount of memory used, as a percent of
     -- the memory bandwidth, as a percent of total bandwidth, both preceding and following the boundary event

NOTE:

Memory bandwidth is obtained by dividing the Run Time of the segment (in Seconds, derived from the Run Time Transfer Function) into the Memory I/O of the segment (in Bytes, derived from the Memory I/O Transfer Function). Thus, PERM assumes that the memory references are evenly spread across the run time of the segment.

   - for each bus resource
     -- the name and Class of the bus
     -- the amount of bus bandwidth utilized, as a percent of total bandwidth available, both preceding and following the boundary event.

NOTE:

Bus bandwidth is obtained by dividing the Run Time of the segment (in Seconds, derived from the Run Time Transfer Function) into the Bus I/O of the segment (in Bytes, derived from the Bus I/O Transfer Function). Thus, PERM assumes that the bus utilization is evenly spread across the run time of the segment.

3. A summary of Processor Activity during the selected time interval, including
   - the name of the Processor
   - the time interval selected by the user
   - the Compute file from which the data was drawn
   - the total time, and percent of time, spent in

-- Application Code segments
-- Operating System segments
-- Join segments

4. A summary of Processor Activity during the entire Epoch interval, including
   - the name of the Processor
   - the time interval selected by the user
   - the Compute file from which the data was drawn
   - the total time, and percent of time, spent in
     -- Application Code segments
     -- Operating System segments
     -- Join segments

When the command is executed, the user will be prompted for the name of the file to which the data is to be written. A full DOS path name, including drive and directory, can be used. If omitted, PERM will use the default directory -- the directory from which PERM was executed. If PERM cannot open the file, and error message is displayed, and PERM waits for further action by the user. If the named file already exists, it will be over-written, and its contents lost.

Once the file name has been entered, the interval start and end times must be entered. These are entered as floating point numbers into a menu provided for this purpose. The default start time is 0.; the default end time is the Epoch length (as specified during System Load definition in the initialization phase of PERM). Both the new start and end times must lie in this interval, and the new end time must be greater than the new start time. Failure to adhere to these rules will generate an error message, and the user can either try again or exit the command, using <ESC>.

Print Event List for a Processor to the Screen (@@DDS)

During the Compute phase of PERM, a record is made of the following events during the course of the simulation:

* the start and end of each segment, including segment run time and resource utilization as derived from the Transfer Functions; and

* the start and end of each Task.

The purpose of this command is to print an ASCII version of the segment boundary events relating to a particular processor, as well as summary information on processor utilization by segment Type (Application Code, Operating System, or waiting to synchronize in a Join), to the screen for review by the user. As usual, the data is scrolled to the screen a page at a time. The user can stop the data output, and return to the menu, by using the <C> key.

The data output by this command consists of the following:

1. A Header that names
   - the name of the Processor being examined
   - the Compute file from which the data was drawn
   - the Processor Ensemble name
   - the Author
   - the System Load Epoch (maximum end time)
   - the start and end times selected by the user when the command was entered

2. For each Segment Boundary Event (the start or end of a segment):
   - the Type of the segment following the event (Application Code, OS, Join)
   - the time of the event
   - the duration of the segment following the event
   - the name and Class of that segment
   - the Task and Task Class of that segment
   - the Thread to which that segment belongs
   - for each memory resource
     -- the name and Class of the memory
     -- the amount of memory used, as a percent of total capacity, both preceding and following the boundary event
     -- the memory bandwidth, as a percent of total bandwidth, both preceding and following the boundary event

NOTE:

Memory bandwidth is obtained by dividing the Run Time of the segment (in Seconds, derived from the Run Time Transfer Function) into the Memory I/O of the segment (in Bytes, derived from the Memory I/O Transfer Function). Thus, PERM assumes that the memory references are evenly spread across the run time of the segment.

   - for each bus resource
     -- the name and Class of the bus
     -- the amount of bus bandwidth utilized, as a percent of total bandwidth available, both preceding and following the boundary event.

NOTE:

Bus bandwidth is obtained by dividing the Run Time of the segment (in Seconds, derived from the Run Time Transfer Function) into the Bus I/O of the segment (in Bytes, derived from the Bus I/O Transfer Function). Thus, PERM assumes that the bus utilization is evenly spread across the run time of the segment.

3. A summary of Processor Activity during the selected time interval, including
   - the name of the Processor
   - the time interval selected by the user
   - the Compute file from which the data was drawn
   - the total time, and percent of time, spent in
     -- Application Code segments
     -- Operating System segments
     -- Join segments

4. A summary of Processor Activity during the entire Epoch interval, including
   - the name of the Processor
   - the time interval selected by the user
   - the Compute file from which the data was drawn
   - the total time, and percent of time, spent in
     -- Application Code segments
     -- Operating System segments
     -- Join segments

The user will be prompted for the interval start and end times. These are entered as floating point numbers into a menu provided for this purpose. The default start time is 0.; the default end time is the Epoch length (as specified during System Load definition in the initialization phase of PERM). Both the new start and end times must lie in this interval, and the new end time must be greater than the new start time. Failure to adhere to these rules will generate an error message, and the user can either try again or exit the command, using <ESC>.

Display the Total Resource Use Profile (@@DR)

The purpose of this command is to display the usage of any system resource -- bus or memory -- over time. The user will be able to select text or graphics mode. If text mode is selected, the data can be routed either to the screen or to a user-selected DOS file. If graphics is selected, the display will be presented on the screen. The user will then have the option of printing it to an attached printer (see the "Select Print Driver" command, <P>, at the top level menu).

The user will specify one of three types of resource utilization to be displayed, corresponding to the three types of resource Transfer Functions:

- Memory Capacity
- Memory I/O bandwidth
- Bus I/O bandwidth

Then the Class and Name of the resource (memory or bus) must be selected. Finally, the time interval must be selected. Details of this process are discussed with the commands on the next lower-level menu.

Internally, PERM divides the total EPOCH interval into 500 distinct bins, and then samples the resource utilization (for every processor) at each bin. This command produces the sum of those values across processors; the Processor/Resource command shows the utilization of the resource by any given processor. The textual display prints out the resource utilization values, as a percent of total resource available, for each time bin. Selecting a sub-interval of time effectively selects a contiguous subset of these bins. In the graphical displays, these values are displayed on a 2-dimensional plot, value-vs-time, over the selected time interval. The plot is automatically scaled to the screen size. In the textual display, the actual bin start and end times, and values, are printed out -- either to the screen, or to a user-selected DOS file. The contents of a textual display are as follows:

1. A header that identifies
   - the resource type: memory capacity, memory I/O bandwidth, or bus bandwidth
   - the instantiated name of the particular resource -- bus or memory -- being examined
   - the start and end times of the time interval
   - the total Size of the resource (drawn from the Class to which the resource belongs)
   - the Compute file from which the data was drawn
   - the Processor Ensemble name
   - the Author

2. For each time bin falling within the selected time interval
   - the number of the interval (1 <= # <= 500)
   - the start and end time of the interval
   - the value of resource utilization in the interval, as a percentage of total resource available

If the graphical display is chosen, the graph of value-vs-time is labeled with the following data:

- the resource type: memory capacity, memory I/O bandwidth, or bus bandwidth
- the instantiated name of the particular resource -- bus or memory -- being examined
- the start and end times of the time interval
- the total Size of the resource (drawn from the Class to which the resource belongs), called the resource Threshold; it corresponds to 100% on the vertical axis
- the Compute file from which the data was drawn
- the Processor Ensemble name

In addition, the axes are labeled with their appropriate units values.

Print a Resource Utilization Profile (to a DOS File) (@@DRF)

Internally, PERM divides the total EPOCH interval into 500 distinct bins, and then samples the resource utilization (for every processor) at each bin. This command produces the sum of those values across processors; the Processor/Resource command shows the utilization of the resource by any given processor. An ASCII version of the resource utilization values, as a percent of total resource available, for each time bin, is written to a user-selected DOS file. Selecting a sub-interval of time effectively selects a contiguous subset of these bins. The contents of a textual file are as follows:

1. A header that identifies
    - the resource type: memory capacity, memory I/O bandwidth, or bus bandwidth
    - the instantiated name of the particular resource -- bus or memory -- being examined
    - the start and end times of the time interval
    - the total Size of the resource (drawn from the Class to which the resource belongs)
    - the Compute file from which the data was drawn
    - the Processor Ensemble name
    - the Author

2. For each time bin falling within the selected time interval
    - the number of the interval (1 <= # <= 500)
    - the start and end time of the interval
    - the value of resource utilization in the interval, as a percentage of total resource available

When the command is executed, the user will be prompted for the name of the file to which the data is to be written. A full DOS path name, including drive and directory, can be used. If omitted, PERM will use the default directory -- the directory from which PERM was executed. If PERM cannot open the file, and error message is displayed, and PERM waits for further action by the user. If the named file already exists, it will be over-written, and its contents lost.

Next, the user must supply the following information: (1) the type of resource to be examined (memory capacity, memory I/O bandwidth, or bus bandwidth); (2) the Class of resource (that is, Memory Class or Bus Class from the Processor Ensemble); (3) the instantiated name of the resource from that Class; and (4) the start and end times of the interval over which data is wanted.

The first three of these are selected from scrolling menus in the usual manner: <Home> and <End> to move the cursor, and <Return> to select.

The interval start and end times are entered as floating point numbers into a menu provided for this purpose. The default start time is 0.; the default end time is the Epoch length (as specified during System Load definition in the initialization phase of PERM). Both the new start and end times must lie in this interval, and the new end time must be greater than the new start time. Failure to adhere to these rules will generate an error message, and the user can either try again or exit the command, using <ESC>.

Print a Resource Utilization Profile (to the Screen) (@@DRS)

Internally, PERM divides the total EPOCH interval into 500 distinct bins, and then samples the resource utilization (for every processor) at each bin. This command produces the sum of those values across processors; the Processor/Resource command shows the utilization of the resource by any given processor. An ASCII version of the resource utilization values, as a percent of total resource available, for each time bin, is written to the screen. As usual, the data is scrolled a page at a time, with the <C> key being available to stop data output and return to the menu. Selecting a sub-interval of time effectively selects a contiguous subset of the time bins for display. The contents of a textual file are as follows:

1. A header that identifies
   - the resource type: memory capacity, memory I/O bandwidth, or bus bandwidth
   - the instantiated name of the particular resource -- bus or memory -- being examined
   - the start and end times of the time interval
   - the total Size of the resource (drawn from the Class to which the resource belongs)
   - the Compute file from which the data was drawn
   - the Processor Ensemble name
   - the Author

2. For each time bin falling within the selected time interval
   - the number of the interval (1 <= # <= 500)
   - the start and end time of the interval
   - the value of resource utilization in the interval, as a percentage of total resource available

The user must supply the following information: (1) the type of resource to be examined (memory capacity, memory I/O bandwidth, or bus bandwidth); (2) the Class of resource (that is, Memory Class or Bus Class from the Processor Ensemble); (3) the instantiated name of the resource from that Class; and (4) the start and end times of the interval over which data is wanted. The first three of these are selected from scrolling menus in the usual manner: <Home> and <End> to move the cursor, and <Return> to select.

The interval start and end times are entered as floating point numbers into a menu provided for this purpose. The default start time is 0.; the default end time is the Epoch length (as specified during System Load definition in the initialization phase of PERM). Both the new start and end times must lie in this interval, and the new end time must be greater than the new start time. Failure to adhere to these rules will generate an error message, and the user can either try again or exit the command, using <ESC>.

Graph a Resource Utilization Profile (@@DRG)

The purpose of this command is to present a graphical display of the usage of any system resource -- bus or memory -- over time.

Internally, PERM divides the total EPOCH interval into 500 distinct bins, and then samples the resource utilization (for every processor) at each bin. This command produces the sum of those values across processors; the Processor/Resource command shows the utilization of the resource by any given processor. Selecting a sub-interval of time effectively selects a contiguous subset of these bins. In the graphical displays, these values are presented in a 2-dimensional plot, value-vs-time, over the selected time interval. The plot is automatically scaled to the screen size. The graph of value-vs-time is labeled with the following data:

- the resource type: memory capacity, memory I/O bandwidth, or bus bandwidth
- the instantiated name of the particular resource -- bus or memory -- being examined
- the start and end times of the time interval
- the total Size of the resource (drawn from the Class to which the resource belongs), called the resource Threshold; it corresponds ..to 100% on the vertical axis
- the Compute file from which the data was drawn
- the Processor Ensemble name

In addition, the axes are labeled with their appropriate units values.

The user must supply the following information: (1) the type of resource to be examined (memory capacity, memory I/O bandwidth, or bus bandwidth); (2) the Class of resource (that is, Memory Class or Bus Class from the Processor Ensemble); (3) the instantiated name of the resource from that Class; and (4) the start and end times of the interval over which data is wanted. The first three of these are selected from scrolling menus in the usual manner: <Home> and <End> to move the cursor, and <Return> to select.

The interval start and end times are entered as floating point numbers into a menu provided for this purpose. The default start time is 0.; the default end time is the Epoch length (as specified during System Load definition in the initialization phase of PERM). Both the new start and end times must lie in this interval, and the new end time must be greater than the new start time. Failure to adhere to these rules will generate an error message, and the user can either try again or exit the command, using <ESC>.

When the graphical display has been drawn on the screen, the user will have the option of printing a hardcopy of it to an attached printer. PERM assumes the printer is mounted on port LPT1, and the print driver must have been select from the top-level Print Driver command so as to match the internal driver to the printer. The <P> key is used to cause the printing to occur.

Display the Resource Use Profile for a Processor (@@DP)

The purpose of this command is to display the usage by a particular processor of any system resource -- bus or memory -- over time. The user will be able to select text or graphics mode. If text mode is selected, the data can be routed either to the screen or to a user-selected DOS file. If graphics is selected, the display will be presented on the screen. The user will then have the option of printing it to an attached printer (see the "Select Print Driver" command, <P>, at the top level menu).

The user will first select the particular Processor to be examined. The user will then specify one of three types of resource utilization to be displayed, corresponding to the three types of resource Transfer Functions:

- Memory Capacity
- Memory I/O bandwidth
- Bus I/O bandwidth

Then the Class and Name of the resource (memory or bus) must be selected. Finally, the time interval must be selected. Details of this process are discussed with the commands on the next lower-level menu.

Internally, PERM divides the total EPOCH interval into 500 distinct bins, and then samples the resource utilization for every processor at each bin. The Total Utilization command <DR> produces the sum of those values across all processors; this command shows the utilization of the resource by the particular, selected processor. The textual display prints out the resource utilization values, as a percent of total resource available, for each time bin. Selecting a sub-interval of time effectively selects a contiguous subset of these bins. In the graphical displays, these values are displayed on a 2-dimensional plot, value-vs-time, over the selected time interval. The plot is automatically scaled to the screen size. In the textual display, the actual bin start and end times, and values, are printed out -- either to the screen, or to a user-selected DOS file. The contents of a textual display are as follows:

1. A header that identifies
   - the processor being examined (that is, its instantiated name
   - the resource type: memory capacity, memory I/O bandwidth, or bus bandwidth
   - the instantiated name of the particular resource -- bus or memory -- being examined
   - the start and end times of the time interval
   - the total Size of the resource (drawn from the Class to which the resource belongs)
   - the Compute file from which the data was drawn
   - the Processor Ensemble name
   - the Author

2. For each time bin falling within the selected time interval
   - the number of the interval (1 <= # <= 500)
   - the start and end times of the interval
   - the value of resource utilization by the processor in the interval, as a percentage of total resource available

If the graphical display is chosen, the graph of value-vs-time is labeled with the following data:

- the processor being examined (that is, its instantiated name)
- the resource type: memory capacity, memory I/O bandwidth, or bus bandwidth
- the instantiated name of the particular resource -- bus or memory -- being examined
- the start and end times of the time interval
- the total Size of the resource (drawn from the Class to which the resource belongs), called the resource Threshold; it corresponds..to 100% on the vertical axis
- the Compute file from which the data was drawn
- the Processor Ensemble name

In addition, the axes are labeled with their appropriate units and values.

Print a Processor/Resource Utilization Profile (to a DOS File) (@@DPF)

Internally, PERM divides the total EPOCH interval into 500 distinct bins, and then samples the resource utilization (for every processor) at each bin. This command shows the utilization of the resource by any given processor; the Total Resource utilization command sums the utilization of the resource across all processors. An ASCII version of the resource utilization values, as a percent of total resource available, for each time bin, is written to a user-selected DOS file. Selecting a sub-interval of time effectively selects a contiguous subset of these bins. The contents of a textual file are as follows:

1. A header that identifies
   - the instantiated name of the processor being examined
   - the resource type: memory capacity, memory I/O bandwidth, or bus bandwidth
   - the instantiated name of the particular resource -- bus or memory -- being examined
   - the start and end times of the time interval
   - the total Size of the resource (drawn from the Class to which the resource belongs)
   - the Compute file from which the data was drawn
   - the Processor Ensemble name
   - the Author

2. For each time bin falling within the selected time interval
   - the number of the interval (1 <= # <= 500)
   - the start and end time of the interval
   - the value of resource utilization in the interval, as a percentage of total resource available

When the command is executed, the user will be prompted for the name of the file to which the data is to be written. A full DOS path name, including drive and directory, can be used. If omitted, PERM will use the default directory -- the directory from which PERM was executed. If PERM cannot open the file, and error message is displayed, and PERM waits for further action by the user. If the named file already exists, it will be over-written, and its contents lost.

The user must next supply the following information: (1) the Class of the processor to be examined; (2) the instantiated name of the processor; (3) the type of resource to be examined (memory capacity, memory I/O bandwidth, or bus bandwidth); (4) the Class of resource (that is, Memory Class or Bus Class from the Processor Ensemble); (5) the instantiated name of the resource from that Class; and (6) the start and end times of the interval over which data is wanted. The first five of these are selected from scrolling menus in the usual manner: <Home> and <End> to move the cursor, and <Return> to select.

The interval start and end times are entered as floating point numbers into a menu provided for this purpose. The default start time is 0.; the default end time is the Epoch length (as specified during System Load definition in the initialization phase of PERM). Both the new start and end times must lie in this interval, and the new end time must be greater than the new start time. Failure to adhere to these rules will generate an error message, and the user can either try again or exit the command, using <ESC>.

Print a Processor/Resource Utilization Profile (to the Screen) (@@DPS)

Internally, PERM divides the total EPOCH interval into 500 distinct bins, and then samples the resource utilization (for every processor) at each bin. This command shows the utilization of the resource by any given processor; the Total Resource utilization command sums the utilization of the resource across all processors. An ASCII version of the resource utilization values, as a percent of total resource available, for each time bin, is written to the screen. As usual, the data is scrolled to the screen a page at a time, with the <C> key available to stop data output and return to the menu. Selecting a sub-interval of time effectively selects a contiguous subset of time bins for display. The contents of a textual display are as follows:

1. A header that identifies
   - the instantiated name of the processor being examined
   - the resource type: memory capacity, memory I/O bandwidth, or bus bandwidth
   - the instantiated name of the particular resource -- bus or memory -- being
   - the start and end times of the time interval
   - the total Size of the resource (drawn from the Class to which the resource belongs)
   - the Compute file from which the data was drawn
   - the Processor Ensemble name
   - the Author

2. For each time bin falling within the selected time interval
   - the number of the interval (1 <= # <= 500)
   - the start and end time of the interval
   - the value of resource utilization in the interval, as a percentage of total resource available

The user must supply the following information: (1) the Class of the processor to be examined; (2) the instantiated name of the processor; (3) the type of resource to be examined (memory capacity, memory I/O bandwidth, or bus bandwidth); (4) the Class of resource (that is, Memory Class or Bus Class from the Processor Ensemble); (5) the instantiated name of the resource from that Class; and (6) the start and end times of the interval over which data is wanted. The first five of these are selected from scrolling menus in the usual manner: <Home> and <End> to move the cursor, and <Return> to select.

The interval start and end times are entered as floating point numbers into a menu provided for this purpose. The default start time is 0.; the default end time is the Epoch length (as specified during System Load definition in the initialization phase of PERM). Both the new start and end times must lie in this interval, and the new end time must be greater than the new start time. Failure to adhere to these rules will generate an error message, and the user can either try again or exit the command, using <ESC>.

Graph a Processors/Resource Utilization Profile (@@DPG)

The purpose of this command is to present a graphical display of the usage by any particular processor of any system resource -- bus or memory -- over time.

Internally, PERM divides the total EPOCH interval into 500 distinct bins, and then samples the resource utilization (for every processor) at each bin. This shows the utilization of the resource by any given processor; the Total Resource command produces the sum of those values across processors. Selecting a sub-interval of time effectively selects a contiguous subset of these bins. In the graphical displays, these values are presented in a 2-dimensional plot, value-vs-time, over the selected time interval. The plot is automatically scaled to the screen size. The graph of value-vs-time is labeled with the following data:

- the processor (that is, its instantiated name) being examined
- the resource type: memory capacity, memory I/O bandwidth, or bus bandwidth
- the instantiated name of the particular resource -- bus or memory -- being examined
- the start and end times of the time interval

- the total Size of the resource (drawn from the Class to which the resource belongs), called the resource Threshold; it corresponds ..to 100% on the vertical axis
- the Compute file from which the data was drawn
- the Processor Ensemble name

In addition, the axes are labeled with their appropriate units values.

The user must supply the following information: (1) the Class of the processor to be examined; (2) the instantiated name of the processor; (3) the type of resource to be examined (memory capacity, memory I/O bandwidth, or bus bandwidth); (4) the Class of resource (that is, Memory Class or Bus Class from the Processor Ensemble); (5) the instantiated name of the resource from that Class; and (6) the start and end times of the interval over which data is wanted. The first five of these are selected from scrolling menus in the usual manner: <Home> and <End> to move the cursor, and <Return> to select.

The interval start and end times are entered as floating point numbers into a menu provided for this purpose. The default start time is 0.; the default end time is the Epoch length (as specified during System Load definition in the initialization phase of PERM). Both the new start and end times must lie in this interval, and the new end time must be greater than the new start time. Failure to adhere to these rules will generate an error message, and the user can either try again or exit the command, using <ESC>.

When the graphical display has been drawn on the screen, the user will have the option of printing a hardcopy of it to an attached printer. PERM assumes the printer is mounted on port LPT1, and the print driver must have been select from the top-level Print Driver command so as to match the internal driver to the printer. The <P> key is used to cause the printing to occur.

3. PERM TEST CASE: AOA/AOSP TRACKING MODEL

3.1 ABSTRACT OF ANALYSIS PROBLEM

To exercise analysis capabilities, a tracking model example was implemented in PERM. This test case was selected not only to demonstrate a realistic Strategic Defense Initiative (SDI) application, but also to stress PERM with a complex system.

The target software for this Track Model design was the Airborne Optical Adjunct (AOA) Mission Data Processor (MDP). The AOA MDP software is divided into several major functions: Angular Rate Smoothing, Candidate Generation Process, Chip Selection, Deferred Object Screening, Handover, Measurement Processing, Navigation Update, Object Screening, Object Sorting, Prediction, Radiometric Discriminant Initialize, Radiometric Discriminant Update, Reference Star Matching, Track Data Management, Trajectory Fitting, Track Initialization, and Track Update.

The MDP functions are allocated to four major subsystems: Measurement Processing; Scan-to-Scan Correlation; Tracking, Discrimination, and Designation; and Navigation, Input/Output, and Control.

The Measurement Processing Subsystem receives Object Sighting Messages. It corrects bias, compensates for aero-optic refractions, and corrects irradiance measurements on these sightings. This subsystem also performs star screening by discriminating stars from other objects. The Scan-to-Scan Correlation Subsystem sorts these object sightings into both azimuth and elevation bins and separates the sightings that are part of established tracks from the ones that are from uncorrelated tracks. The established track sightings are sent on to the Tracking, Discrimination, and Designation Subsystem. The uncorrelated sightings are compared to previous frames of sightings to attempt to form candidate ballistic trajectories. These candidate tracks are also sent to Tracking, Discrimination, and Designation.

The Tracking, Discrimination, and Designation Subsystem uses navigation data, candidate track messages, and object sighting data to initialize, validate, and update precision tracks. This subsystem uses correlated object sightings to predict the position and velocity of the tracks at both handover and impact point. Lethality is also estimated for each track.

The Navigation, Input/Output, and Control Subsystem performs atmospheric refraction compensation, reference star matching, navigation update, module selection, scan control, handover buffer, and external MDP operational interfaces.

For more detailed information about the AOA MDP flight software algorithms, reference the Mission Data Processor Algorithm Design Document, D461-10282, Boeing Aerospace Corporation, 30 January 1987, classified SECRET.

The AOA MDP functions depend heavily on the data passed between them. This intricate data flow is portrayed in Figure 3-1. (This chart is from Teledyne Brown Engineering Technical Letter "MDP Architecture Informal Class", KD87-AOA-HA-3-2-0114, N. E. Reed, 30 June 1987.)

FUNCTIONS

ARS	Angular Rate Smoothing
CGP	Candidate Generation Process
CS	Chip Selection
DES	Designation
DOSC	Deferred Object Screening
HO	Handover
MP	Measurement Processing
NAV	Navigation Update
OSC	Object Screening
OSO	Object Sorting
PR	Prediction
RDI	Radiometric Discriminant Initialize
RDU	Radiometric Discriminant Update
RSM	Reference Star Matching
SC	Scan Control
TDM	Track Data Management
TF	Trajectory Fitting
TI	Track Initialization
TU	Track Update

DATA

CTM	Candidate Track Message
DMSM	Deferred Missed Sighting Msg
DPWF	Deferred Predicted Window File
DTUM	Deferred Track Update Message
ELCOR	Elevation Correction Message
HTSM	Handover Track Status Message
INA	Initiator Assignment Data
INR	Initiator Request
MSM	Missed Sighting Message
NHF	Navigation History File
NTM	New Track Message
OIF	Object Irradiance File
OSDF	Object State Data File
OSM	Object Sighting Message
OSR	Object Sighting Record
OTF	Object Track File
PCTM	Partial Candidate Track Message
PDTUM	Partial Deferred Track Update Msg
PRH	Prediction to Handover Message
PWF	Predicted Window File
SCL	Scan Control Limits
SSM	Star Sighting Message
SSOS	Sensor Start of Scan Message
TAM	Track Acceptance Message
TARM	Track Accept/Reject Message
TDRM	Track Dropped Message
TIM	Track Initiation Message
TUM	Track Update Message
USM	Uncorrelated Sighting Message

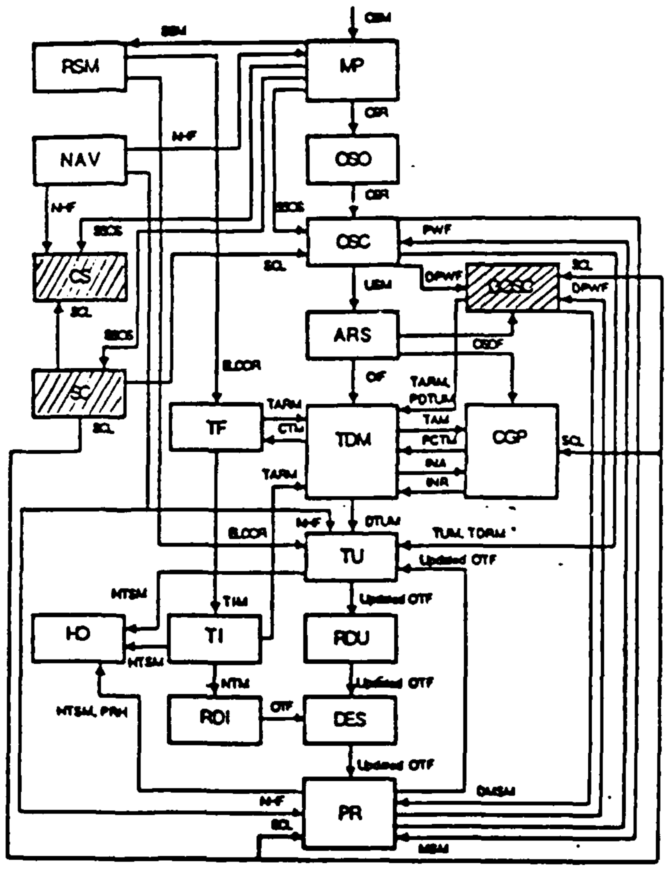


Figure 3-1 MDP Data Flow

The MDP flight software is distributed on nine processing nodes on the AOA hardware, shown in Figure 3-2. (This diagram is based on Boeing Aerospace Corporation Technical Letter "Flight Software Architecture", 2-2985-87JS-019, R. Schroder, 6 February 1987.) The first node hosts the Object Sorting and Measurement Processing functions. Node 2 hosts Object Screening, Angular Rate Smoothing, Track Data Management, and Predicted Window File Sort. Nodes 3-8 are responsible for Candidate Track Selection and Tracking, Discrimination, and Designation. Node 9 is the Navigation, Input/Output, and Control Node responsible for Navigation Update, Reference Star Matching, and Handover Communication.

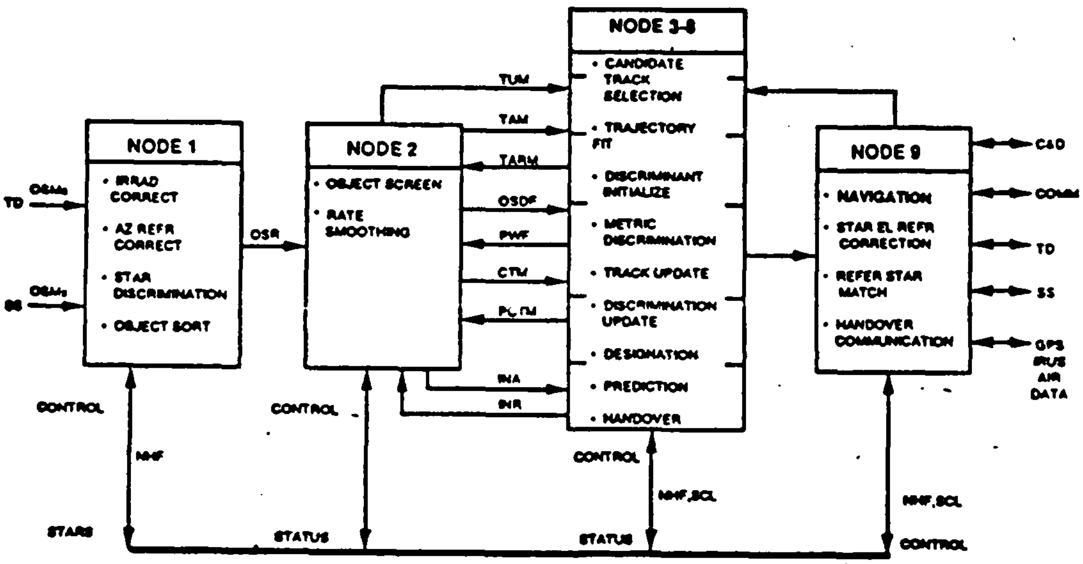


Figure 3-2 AOA MDP Software Configuration

The target hardware for the Track Model design was the Advanced Onboard Signal Processor (AOSP). The AOSP can be viewed at four different levels: CPU, node, platform and system.

The AOSP CPU is a processor chip that is based on the 1750A standard design, a RISC architecture using a 16-bit word, as the heart of the Array Computing Element (ACE). Clock speed depends on the manufacturing implementation. Two contractors have developed alternate versions, and performance has steadily been improved in the several years of developments. In this report, the confirmed processing speed attained by the Raytheon version has been used.

The ACEs are combined into complexes designated as nodes. These include several CPU/ACE sets, memory modules, memory control units (DMA), processor accelerators for selected functions (e.g. vector arithmetic), and local busses. Figure 3-3 shows a somewhat simplified schematic of an AOSP node, as designed for BSTS application. The DMAs, while actually present in the BSTS hardware, have been omitted in this diagram as a simplification. Further simplification occurs in Figure 3-4, where the separate memory units are coalesced into one, assumed homogeneous. The accelerators have also been suppressed in this diagram, as have interfaces to external busses. The latter will reappear in the next figure. These simplifications are consistent with the intention to model the AOSP with PERM, which provides intermediate fidelity.

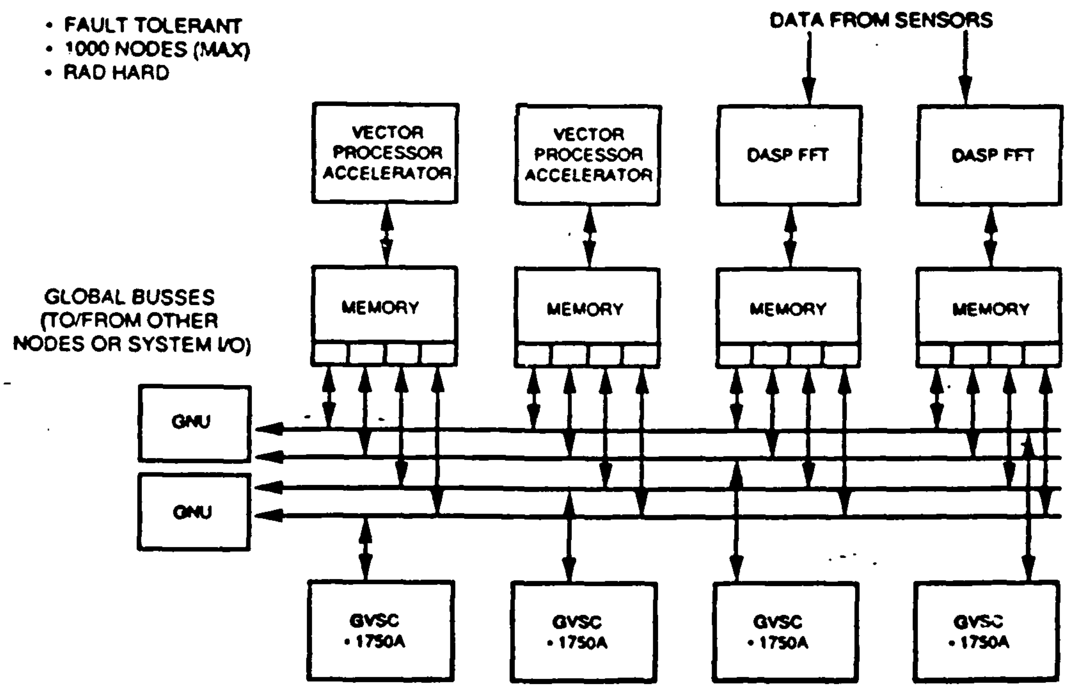
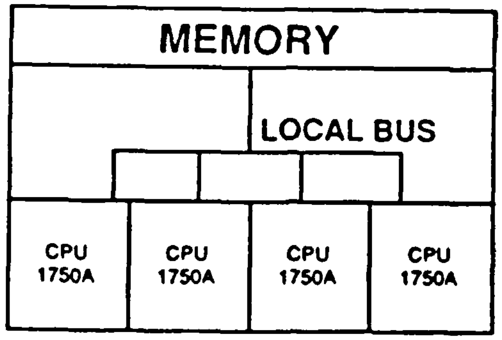


Figure 3-3 AOSP BSTS/ADOP Multi-process



- FOUR INDEPENDENT PROCESSORS PER NODE
- MEMORY ACCESS SCHEME - ARBITRATED CONTENTION
- BUS INTERFACES OMITTED

Figure 3-4 Simplified AOSP Node.

On each platform, the multi-processor nodes of Figures 3-3 or 3-4 are replicated and interconnected by what is termed a planar array architecture. In such an architecture, each node is directly bus-connected to four others in a pattern that is conveniently represented on a plane with n^2 nodes. The four busses at each node are identified respectively as horizontal, vertical, ascending diagonal, and descending diagonal. In Figure 3-5, a planar array of 9 nodes ($= 3^2$) is depicted with the 12 separate busses. Other examples of planar nodal AOSP arrays have been built of sizes 4^2 and 5^2. The sample problem treated in this report requires only five nodes with two or three CPUs per node. The details appear in the next section.

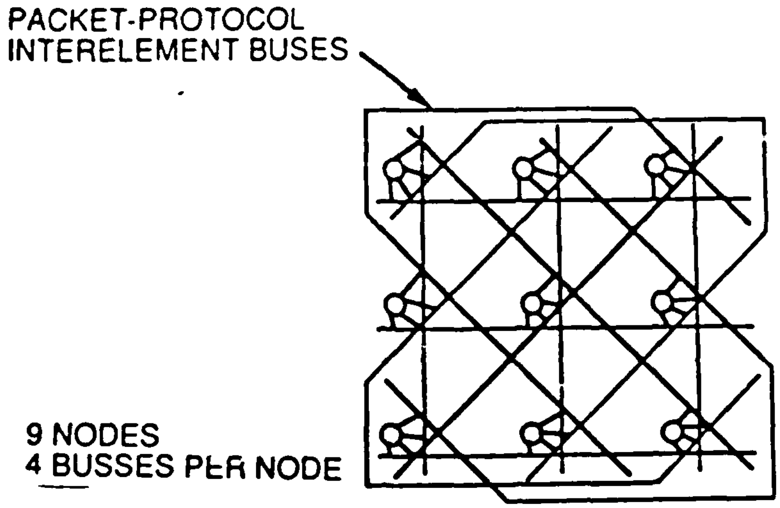


Figure 3-5 AOSP Hardware Configuration Overview

In a Strategic Defense System application, there would be several AOSP-equipped platforms, physically widely separated, e.g. on distinct satellites. They would be interconnected by RF communications links (or possibly lasercomm). Their various functions would be mutually supporting for a wide variety of battle management applications, of which threat object tracking and correlation is just one. The PERM model is well suited to the analysis of such a complex situation, but in this initial investigation the study extended only to single platform level processing involving several nodes and many processors.

To represent the AOSP hardware bandwidth and memory storage, parameters were input into PERM. The effective bus bandwidth of the Track Model is 12 megabytes/second; the memory bandwidth is 16 megabytes/second; and the storage capacity on each nodal memory is eight megabytes.

The AOA software mapped onto the AOSP hardware proved to be an interesting and stressing example to model in PERM. The following pages describe the design, implementation, and results of this track model.

3.2 MODEL DESIGN

Several assumptions were made before implementing the Track Model in PERM. These assumptions were made for various reasons, many times for simplicity. For example, only two of the six track nodes were modeled in order to limit the test case to a reasonable analysis effort. Also, AOA hardware specific functions were not modeled, such as Scan Control and Chip Selection. Deferred Object Screening was not modeled since only a few objects are deferred and the process is difficult to model. These functions that were not modeled are shaded in the diagram shown in Figure 3-1. External input and output were also not modeled in order to narrow the scope of the analysis.

Some assumptions were made to ensure modeling accuracy. For instance, the number of objects in the system is assumed to be greater than six objects. This assumption is necessary because some of the Lag Segments (segments inserted to more accurately model pipelining) process a constant number of objects.

Another assumption is that the steady state of the system is reached with a simulation time greater than 100 seconds. At this time, a certain number of tracks will be in the system in an "update mode." This assumption was needed to realistically model the tracking process. For example, Object Screening decrements the Predicted Window File by the previous window's total accepted tracks before the Predicted Window File Sort increments this same file with the new tracks.

At least one modeling decision was made to ensure that the example model remained unclassified. For example, the Mission Initialized Data Base is not modeled since its size and distribution across the nodal memories is classified.

Another assumption was made because of limited resources. The allocation of the MDP software to the AOSP hardware was assumed not to increase the resource contention delays that exist with the MDP software on the AOA hardware. Although the AOA software to AOSP hardware mapping is a reasonable scheme, it is probably not optimal. Since PFRM is a feasibility tool and does not attempt to optimize, the allocation scheme was not investigated in enough detail to prove or disprove optimality.

Available data influenced another decision. Much of the timing data used in deriving the runtime transfer equations was extracted from an available timing study (referenced in Appendix B). Since this data included Direct Memory Access, scheduling, and bus access and contention, these costs are also included in the runtime equations.

Another decision was to represent the entire tracking function as a single task. This decision was made, not only for design simplification, but also to prepare for later extensions of the model. For example, multi-task applications could then be easily constructed by modeling multiple attack waves as replications of this single task.

The single track task was modeled as 13 threads divided into 46 application segments. Fourteen join segments were also added to model the thread dependencies. The inter-dependencies of the segments and threads are shown in Figure 3-6. Arrows in this directed graph show the dependencies. Wherever a segment in this graph has an arrow pointing to it, it must wait to start processing until its predecessor segment is finished.

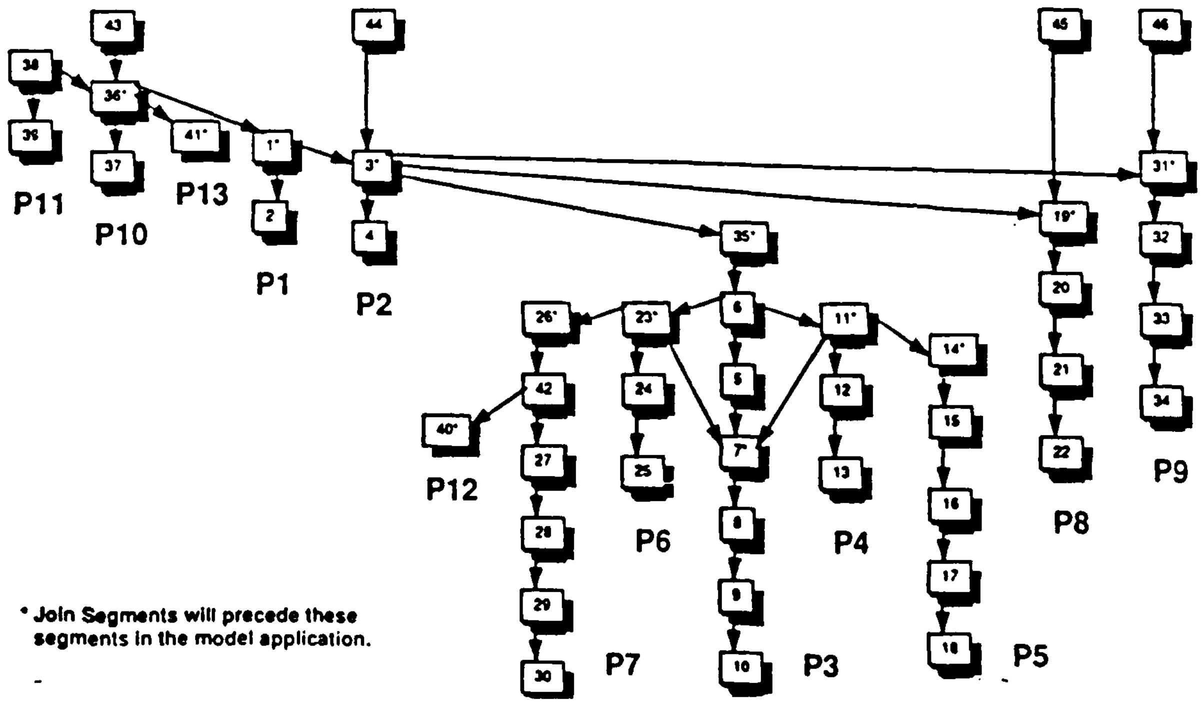


Figure 3-6 Track Task Thread Dependencies

At the start time of a simulation, segments 38, 43, 44, 45, and 46 begin simultaneously. These segments initialize each nodal memory and require no runtime. They utilize memory for the software instruction sets and database steady state sizes. Once these segments finish, still at the start time, segments 36, 3, 19, and 31 begin processing and pipelining begins. Lag Segments (1, 3, 11, 23, 36, and 42) are included where requirements dictate that a certain number of objects must be processed before pipelining can continue.

The allocation of the application segments to the threads is described in Figure 3-7. Each thread is resident on a single processor. Unique threads are Object Screening, Object Sorting, Angular Rate Smoothing/Track Data Management, Measurement Processing, Navigation Update, Handover, and Reference Star Matching. The remaining threads, residing on the tracking nodes, are replicated. These replicated threads are Candidate Generation Process, Track Initialization, and Track Update.

Object Sorting (OSO), P1

1 Object Sorting Lag
2 Object Sorting

Object Screening (OSC), P2

44 Initialization Node 2
3 Object Screening Lag
4 Object Screening

Angular Rate Smoothing/Track Data Management (ARS/TDM), P3

35 Angular Rate Smoothing Lag
6 TDM Initial Initiator Loading
5 Angular Rate Smoothing
7 TDM Build Candidate Track Message
8 TDM Remaining Initiator Loading
9 TDM Track Accept/Reject Message Handler
10 Predicted Window File Sort

Candidate Generation Process (CGP), P4 & P6

11 & 23 Candidate Generation Process Lag
12 & 24 Candidate Generation Process
13 & 25 CGP Track Accept Message Handler

Measurement Processing (MP), P10

43 Initialization Node 1
36 Measurement Processing Lag
37 Measurement Processing

Track Initialization (TI), P5 & P7

14 & 26 Track Fitting
42 Track Initialization Lag
15 & 27 Track Initialization
16 & 28 Radiometric Discriminant Initialization
17 & 29 Designation
18 & 30 Prediction

Track Update (TU), P8 & P9

45 & 46 Initialization Nodes 4 & 5
19 & 31 Track Update
20 & 32 Radiometric Discriminant Update
21 & 33 Designation
22 & 34 Prediction

Navigation Update (NAV), P11

38 Initialization Node 3
39 Navigation Update

Handover (HO), P12

40 Handover

Reference Star Matching (RSM), P13

41 Reference Star Matching

Figure 3-7 Segment to Thread Descriptions

The 13 processors are allocated on five processing nodes, shown in Figure 3-8. The Object Sorting Node hosts both the Measurement Processing Thread and the Object Sorting Thread. The Object Screening Node hosts the Object Screening Thread and the Angular Rate Smoothing/Track Data Management Thread. The third node, Navigation, Input/Output, and Control, hosts the Navigation Update Thread, the Handover Thread, and the Reference Star Matching Thread. Nodes 4 and 5, Tracking Nodes, host the replicated threads of Candidate Generation Process, Track Initialization, and Track Update. Seven unique busses are required to handle the data traffic.

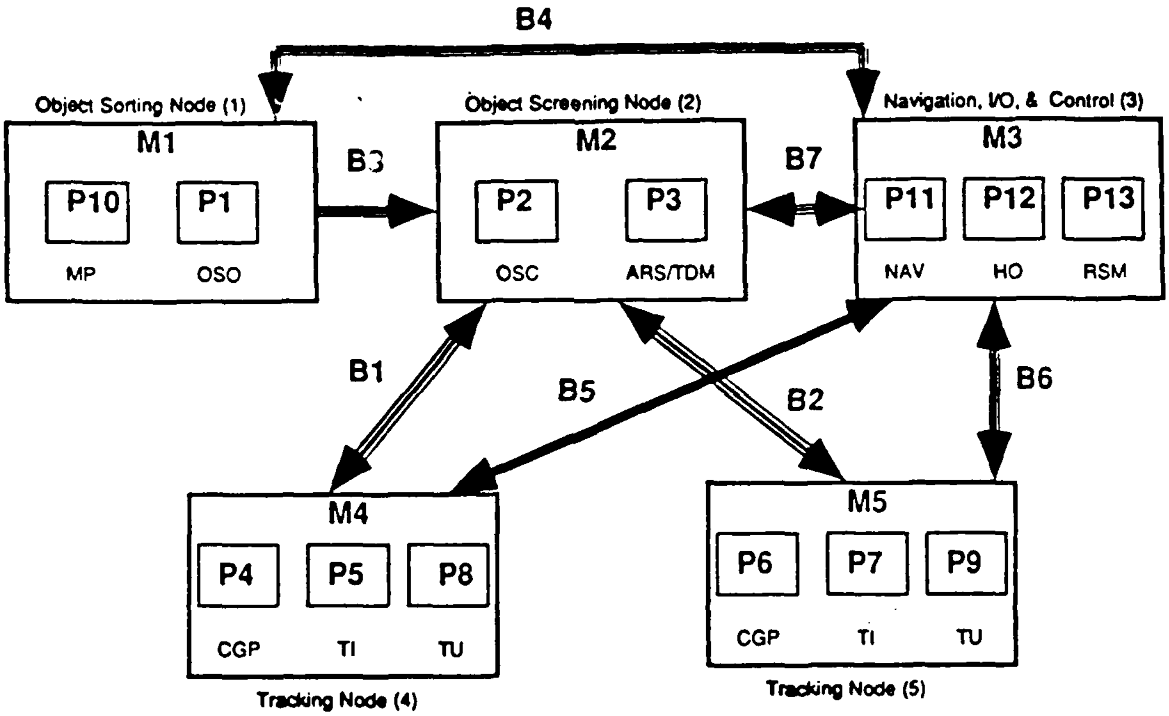


Figure 3-8 PERM Allocation of Software to Hardware

Local and shared memories, designated by M1-M5, reside on each processing node. The primary Track Data Files are allocated to the five nodal memories by the mapping shown in Figure 3-9. The Measurement Processing Nodal Memory resides on M1, the Object Sorting Node's memory. The Object Irradiance File, the Predicted Window File, and the Failed Track List File all reside on M2, the Object Screening Node. The Navigational History File is located on M3, the Navigation, Input/Output, and Control Node's resident memory. The Tracking Nodes host separate copies of the Object State Data File and the Object track File.

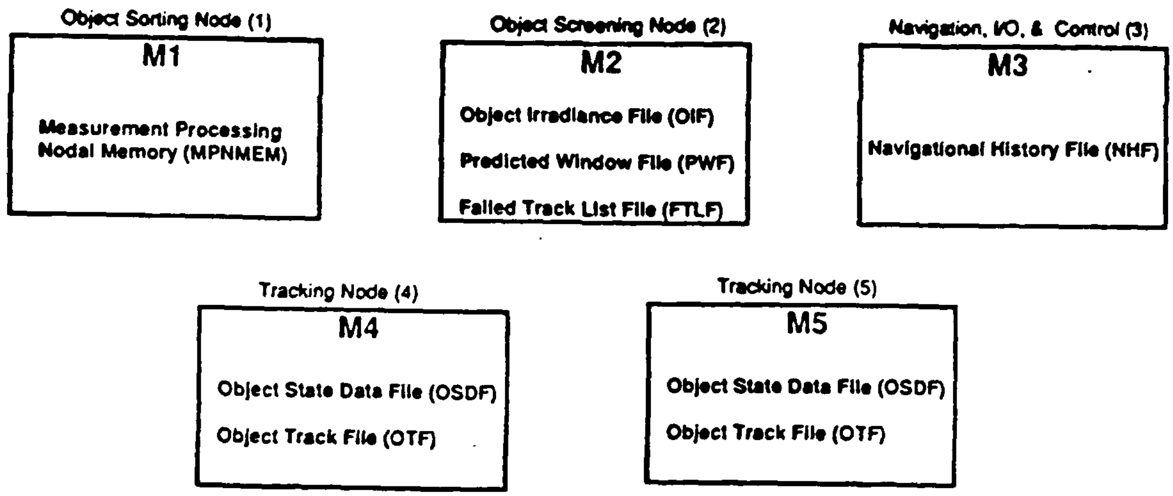


Figure 3-9 Track Data Set Allocation

Runtime, Memory Access, Memory Usage, and Bus Access equations were derived for each of the 46 application segments. These transfer equations are described in detail in Appendix B. Each equation is first given as a function of several defined parameters and then simplified to a function of a single parameter N, the number of objects to track. Runtime is expressed as a single equation; however, Memory Usage, Memory Access, and Bus Access are expressed as sets of equations. Memory Usage and Memory Access equations are divided into five different equations representing the five separate local nodal memories, M1-M5. Similarly, Bus Access equations are divided into five sets representing the seven busses, B1-B7.

3.3 IMPLEMENTATION OF MODEL DESIGN INTO PERM

In implementing the Track Model design into PERM, classes for the processors, memories, and busses were first defined and later instantiated. These class names and instantiations are listed in Figure 3-10. By defining only four classes of processors, all 13 processors were instantiated. Four classes of memories were defined, with five instantiations; and five bus classes were created with seven instantiations. Creating classes first and then using these definitions for several instantiations reduced redundant data entry. These classes and instantiations defined the AOSP hardware, at least the portion of it needed for this application.

CLASSES	INSTANTIATIONS
Processor	
SORTERS	P1 & P10
SCREENERS	P2 & P3
NAVIGATIONAL CONTROLLERS	P11, P12, & P13
TRACKERS	P4, P5, P6, P7, P8, & P9
Memory	
SORT MEMORY	M1
SCREEN MEMORY	M2
NIOC MEMORY	M3
TRACK MEMORY	M4 & M5
Bus	
SCREEN WITH TRACK	B1 & B2
SORT TO SCREEN	B3
SORT WITH NIOC	B4
TRACK WITH NIOC	B5 & B6
SCREEN WITH NIOC	B7

Figure 3-10 PERM Track Model Classes

To represent the AOA tracking software in PERM, segment class descriptions were then built for all the Track Model segments by using the coefficients defined for the transfer equations. (The transfer equations are defined in detail, along with the parameters used in their derivation, in Appendix B.) A partial Track Model segment class description is shown in Figure 3-11. This segment class, OBJECT SORTING, runs on type SORTERS processor class. Its type is an application segment, and it is instantiated only once. The runtime transfer function was entered into the database by simply editing the coefficients. Since the equation was quadratically dependent upon N, three fields were provided. All other coefficients defaulted to zero.

```
*** Segment Class Name: OBJECT SORTING
    Target Processor Class: SORTERS
    Number of Instantiations: 1
    Segment Class Type: Application Code

**** Transfer Functions List:
     N = Data Set Size:
     R = Data Set Size Reduction Factor:
     M = R * N:
     F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M **2) + (L1 + (L2 * M) LOG2(M))

     Run Time Transfer Function Coefficients:

     R = 1.000000E+00   Q1 = 1.900000E-05 L1 = 0.000000E+00
                        Q2 = 9.900000E-05 L2 = 0.000000E+00
                        Q3 = 1.800000E-07
```

Figure 3-11 Track Model Segment Class Description Part 1

The OBJECT SORTING segment class description is continued in Figure 3-12. The SORTER class of processor has access only to one memory, from the SORT MEMORY class. This instantiation was named V_Sort_Memory. Therefore, only one transfer function was defined for memory space requirements and only one transfer function was defined for memory I/O requirements. Since this processor type had access to two busses, two separate transfer equations were defined. This particular segment did not interact with the Navigation, Input/Output, and Control Node, so the coefficients for the V_NIOC_I/O bus were left as zeros.

Memory Space Requirements Transfer Function Coefficients:

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 1.900000E-05 L1 = 0.000000E+00
Q2 = 4.370000E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coefficients:

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 4.370000E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coefficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00 Q1 = 1.760000E+02 L1 = 0.000000E+00
Q2 = 4.370000E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: SORT WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Figure 3-12 Track Model Segment Class Description Part 2

Next, the thirteen threads were built by instantiating segment classes and appending them together. A description of an example thread is shown in Figure 3-13. This replicated Candidate Generation Process thread, "THD_4: CGP1", runs on processor P4 of type TRACKERS. This thread is made up of four segments. Its application segments are Candidate Generation Process Lag, Candidate Generation Process, and Candidate Generation Process Track Accept Message Handler. The join segment is included at the top of the thread to model the dependency of this thread on Segment 6 of the thread, "THD_3: ARS/TDM". This thread was created in the Task Class database by copying the other replicated Candidate Generation Process thread and editing a few fields to change the segment names and the join dependency.

```
*** Thread Name: THD_4: CGP1
    Target Processor: P4
    Target Processor Class: TRACKERS

****Segment List:

      Segment Name: Segment 11 Join
      Segment Class: CGP LAG JOIN
      Segment Type: Join
      Predecessor Segment: Segment 6
      Predecessor Thread: THD_3: ARS/TDM

      Segment Name: Segment 11
      Segment Class: CANDIDATE GENERATION PROCESS LAG
      Segment Type: Application Code

      Segment Name: Segment 12
      Segment Class: CANDIDATE GENERATION PROCESS
      Segment Type: Application Code

      Segment Name: Segment 13
      Segment Class: CGP TRACK ACCEPT MESSAGE HANDLER
      Segment Type: Application Code
```

Figure 3-13 Track Model Thread Description

A complete description of the information in the databases for the Track Model is found in Appendix C. This is a computer-generated printout of the load definition database.

3.4 VERIFICATION APPROACH

The primary purpose for implementing the Track Model design into PERM was to verify the PERM software. This verification process involved seven steps: (1) preparing the test case, (2) making hand calculations, (3) performing PERM System Definition, (4) comparing input values with the database, (5) performing PERM Compute, (6) performing PERM Display, and (7) comparing hand-calculations with the Display outputs. Figure 3-14 depicts this process.

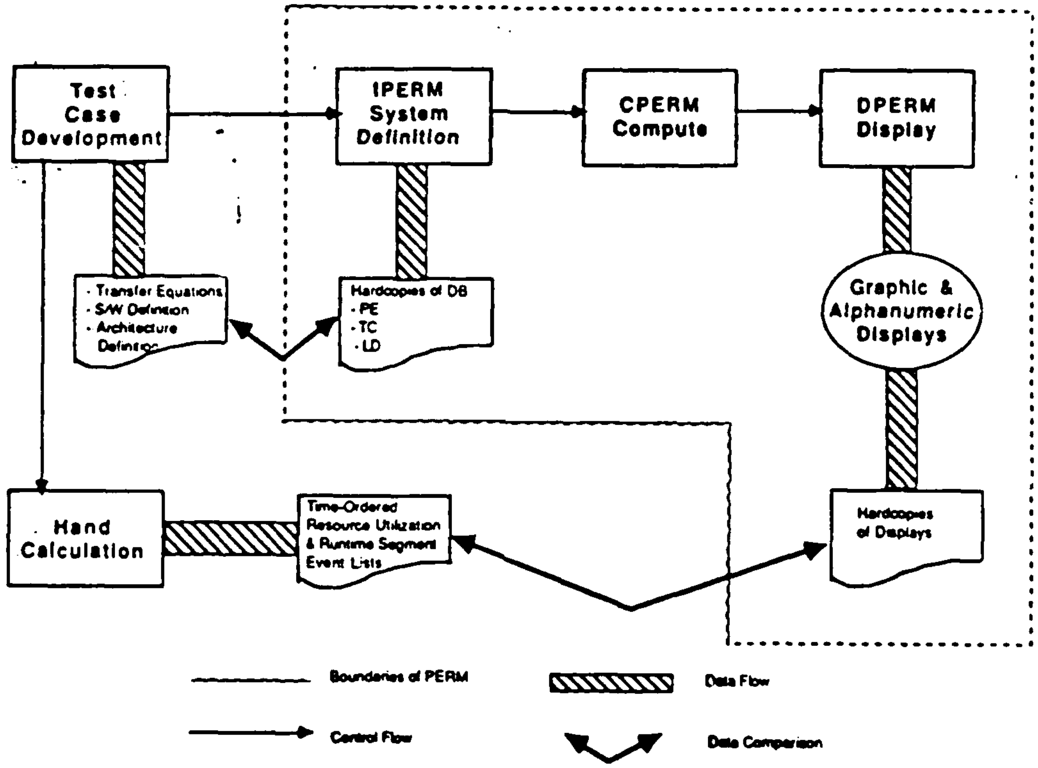


Figure 3-14 PERM Verification Approach

First, the test case was prepared by defining the model as described in section 3.2 and by assigning variable and class names for use in PERM as detailed in section 3.3.

Next, hand calculations were made by assigning a constant value to the parameter N and solving for all the Runtime, Memory Access, Memory Usage, and Bus Access transfer equations. In the hand calculations, beginning runtime for each segment was determined by summing the runtime values of all its predecessor segments. Runtime for each thread was calculated by subtracting the beginning time of the first segment from the end time of the last segment. Resource utilization was calculated by summing all the segment utilization values for each resource within a specific time interval. Since hand calculations for resource use were lengthy and tedious, complete event lists were not generated for the entire model. Instead, time-ordered event tables were calculated for a representative type of each resource. The resources evaluated were bus B1, memory M2, and processor P3.

The PERM system was defined by executing the IPERM program and manually entering the Track Model data. Processor Ensemble Definition (PE); Software Component, or Task Class Definition (TC); and System Load Definition (LD) databases were created.

The PE, TC, and LD files were then converted to ASCII files using the PERM PRINT option and printed using the DOS Print Command. These hardcopies were compared to the input data to check for input errors and to test the validity of the database build generation. Next, the PERM Compute feature was exercised by running the CPERM program. The input file for this Compute operation was the LD file. Event and Index files were generated as output.

The output files from CPERM were used as inputs to the Display program, DPERM. Data-reduced textual and graphical displays were generated for analysis. These displays included an Event List, a textual listing of events within a user-specified interval; a Detailed Processor Profile, a textual listing of only those events involving a user-specified processor; a Resource Use Profile, a textual and graphical summary of the percent utilization of a resource over a particular interval; and a Processor Resource Profile, a textual and graphical summary of a Resource Use Profile for a specific processor.

The major verification effort involved comparing the hand calculated data with the display outputs. Since none of the PERM-generated outputs were identical in form to the hand calculated lists, a combination of Display outputs were analyzed. Segments that used a target resource were extracted from the Event List and compared for validity against the hand calculations. Also, the Resource Use Profile was used to check the percent utilization of this target resource over the specified time interval.

3.5 PERM VERIFICATION RESULTS

The first step in analyzing the test case results was verifying the PERM software by comparing the expected values from hand calculations with the PERM output files from representative resources. The resources evaluated were bus B1, memory M2, and processor P3. Hand calculations were made for the segments running on processor P3 by assigning a constant value of 50 to the parameter N and solving for all the Runtime, Memory Access, Memory Usage, and Bus Access transfer equations. Beginning runtime for each segment was determined by summing the runtime values of all its predecessor segments. Maximum resource utilization percentages were calculated by dividing the values from the resource equations by the maximum resource utilization values allowed during the segments' duration.

As shown in Figure 3-15, the hand-calculated values were very close to the values in the PERM event listing for processor P3. (The complete PERM-generated event list for processor P3 is in Appendix D.) The slight differences in the two tables of values can be attributed to differences in the arithmetic precision used in the calculations and in the order of the calculation operations. The hand-calculation table revealed a little more activity on P3 than the Event Listing showed. The major area of difference was in the memory capacity usage. The PERM event listing showed no usage on each of the nodal memories accessible to P3; however, the hand calculations showed that although the usage was insignificant, there was some activity. The difference in these utilizations was attributed to an improper order of operations in the PERM prototype and has since been corrected.

3.6 TRACK MODEL TEST CASE RESULTS

Three major areas of resource usage were evaluated in the Track Model test case analysis: Memory Capacity, Memory Input/Output Bandwidth, and Bus Bandwidth. The most significant results from each of these areas is discussed in the following paragraphs.

With only 50 objects in the system, the percentage each nodal memory used was approximately zero over the duration of the simulation run. To analyze the nodal memory fluctuation over time, the memory capacity was evaluated for a more realistic threat scenario of 1,000 objects in the system.

PERM Event Listing for P3

Segment	Start	Duration	Max % Usage Bus Bandwidth				Max % Usage Memory Bandwidth			Max % Usage Memory Capacity		
			B1	B2	B3	B7	M2	M4	M5	M2	M4	M5
35	.00578797	.00029000	0	0	0	0	4	4	4	0	0	0
6	.00607797	.00060000	0	0	0	0	0	0	0	0	0	0
5	.00607797	.00546500	0	0	0	0	34	4	4	0	0	0
7	.25480000	.00171500	6	6	0	0	17	0	0	0	0	0
8	.25651500	.00048000	0	0	0	0	0	0	0	0	0	0
9	.25699500	.00440500	0	0	0	0	48	0	0	0	0	0
10	.26140000	.00045000	0	0	0	0	104	0	0	0	0	0

Hand Calculation of Events for P3

Segment	Start	Duration	Max % Usage Bus Bandwidth				Max % Usage Memory Bandwidth			Max % Usage Memory Capacity		
			B1	B2	B3	B7	M2	M4	M5	M2	M4	M5
35	.005788	.000290	0.99	0.99	0	0	4.14	4.14	4.14	0.0024	0.0024	0.0024
6	.006078	.000600	0.08	0.08	0	0	0.00	0.00	0.00	0.0024	0.0000	0.0000
5	.006078	.005465	1.00	1.00	0	0	34.03	4.17	4.17	0.0480	0.0480	0.0480
7	.254801	.001715	8.12	8.12	0	0	16.94	0.00	0.00	3.4000	0.0000	0.0000
8	.256516	.000480	0.08	0.08	0	0	0.00	0.00	0.00	3.4000	0.0000	0.0000
9	.256996	.004405	4.95	4.95	0	0	48.85	0.00	0.00	3.3700	0.0000	0.0000
10	.261401	.000450	0.00	0.00	0	0	104.64	0.00	0.00	3.3700	0.0000	0.0000

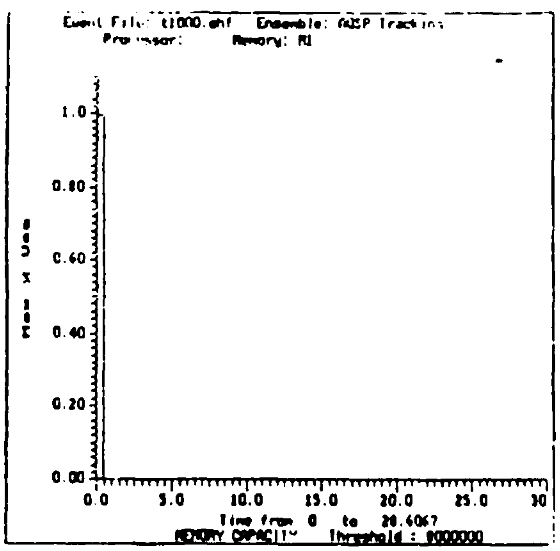
N = 50 objects in the system **Memory Bandwidth = 16000000 bytes/sec. per nodal memory**
Bus Bandwidth = 12000000 bytes/sec. per bus **Memory Capacity = 8000000 bytes per nodal memory**

Figure 3-15 PERM Verification Analysis of Event Listing for P3

The Object Sorting and Object Screening Nodal Memory graphs in Figure 3-16 show that even at 1,000 objects in the system, the utilization was extremely low. The Object Sorting Nodal Memory had some activity near the start of the run, when object sightings were loaded into its buffer and the Measurement Processing Nodal Memory File was initialized for steady state. The Object Screening Nodal Memory maximum use was during the time that the Object Screening Segment 4 was running.

N = 1000 Objects in the System

OBJECT SORTING NODAL MEMORY (M1)



OBJECT SCREENING NODAL MEMORY (M2)

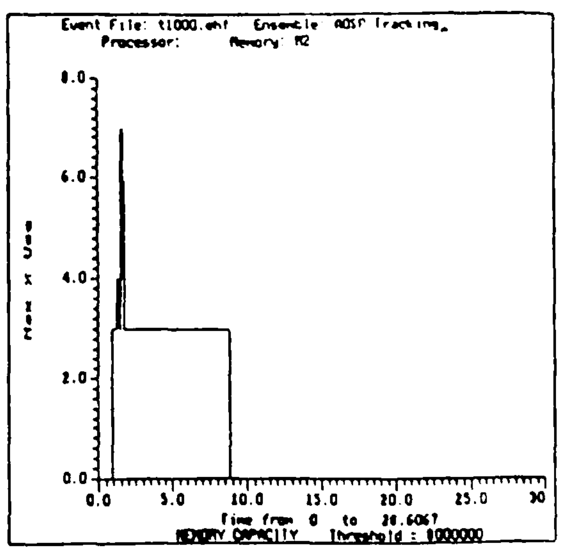


Figure 3-16 Track Model Memory Capacity Analysis Part 1

Since the current PERM version evaluates memory capacity only during each segment's run instead of as a cumulative total, modeling the database fluctuations was somewhat awkward. To model databases, the analyst must know the order that the segments execute and then derive each memory usage equation as a function of what is already in memory. Also, "filler segments" have to be included between timing gaps in segment runs to keep the memory size from dropping when the database was not being decremented. For example, a filler segment was needed for the timing gap between 23 and 24 seconds in the Tracking Nodal Memory chart shown in Figure 3-17. Fidelity of database modeling is also lost when the analyst must make assumptions on how to model parallel segments accessing the same memory. A needed enhancement to PERM is to include an option to model memory capacity by using delta values for each segment that would change the total memory used on a node. This would also allow the analyst to set up segments that permanently allocated memory.

N = 1000 Objects In the System

NIOC NODAL MEMORY (M3) TRACKING NODAL MEMORY (M4 & M5)

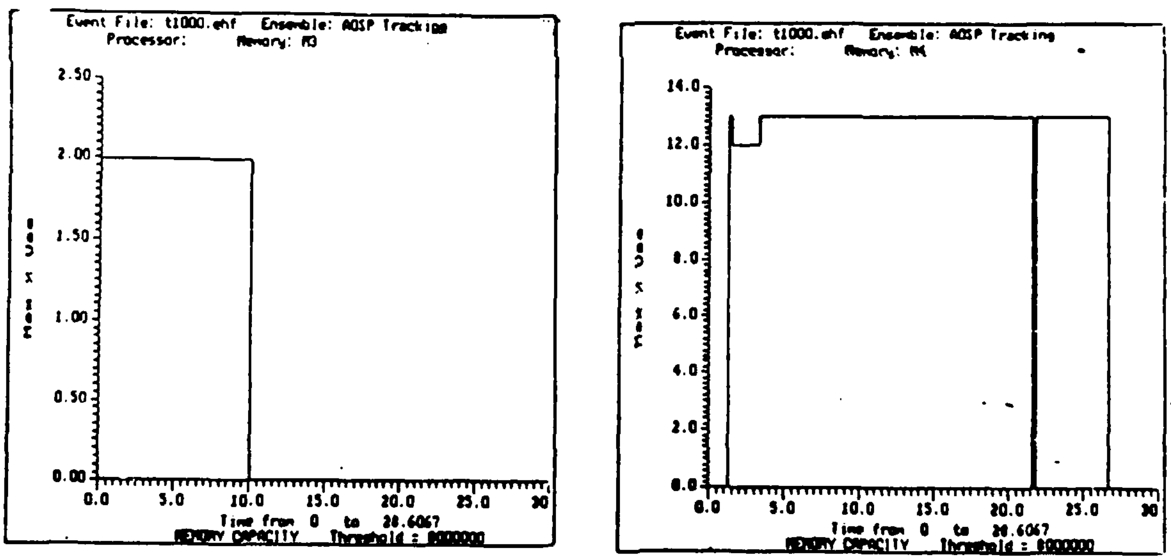


Figure 3-17 Track Model Memory Capacity Analysis Part 2

The Navigation, Input/Output, and Control (NIOC) Nodal Memory, shown in Figure 3-17, also had extremely low utilization, even for 1,000 objects. Since the nodal memory usage is constant (based on the Navigational History File steady state size, the buffer size, and the instruction set size), this maximum percent utilization value should have been the same for 50 objects in the system. This difference is probably due to round-off errors in PERM calculations when using very small values for transfer equations. Also, the percent utilization of the NIOC Nodal Memory should be constant over the entire run, instead of dropping off to zero at the end of the last segment using M3. Again, a "filler segment" should have been included to model the constant memory over the duration of the run.

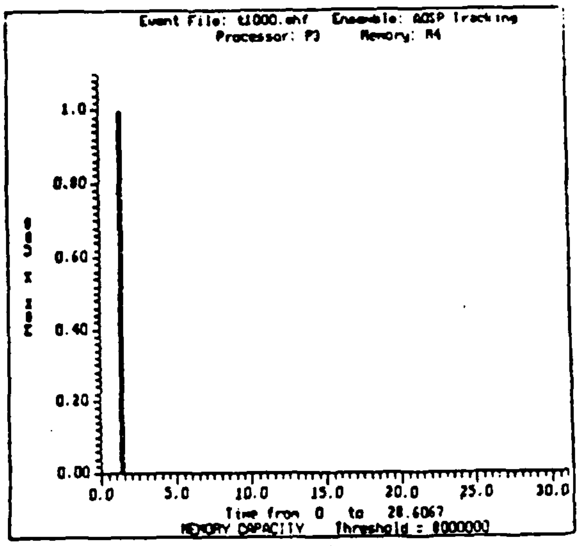
The Tracking Nodal Memory graphs, also shown in Figure 3-17, showed the most activity of the simulation run. Since they are replicated, both Tracking Nodal Memories would be expected to have similar resource utilizations; in this Track Model they were modeled to have the exact same utilizations.

Using PERM's Processor-Resource Utilization Display capabilities, the data on a Tracking Nodal Memory, M4, was reduced to specific processor utilization. This allowed an in-depth analysis of the Tracking Nodal Memory usage at 1,000 objects in the system. The processors using M4 were P3, P5, and P8. Figure 3-18 shows Processor P3's graph of its utilization of M4's capacity.The spike in this graph was caused by Angular Rate Smoothing incrementing the Object State Data File. The Track Initialization Thread, located on P5, is responsible for incrementing the Object Track File; therefore, it was the primary contributor to the usage of M4 toward the end of the simulation run. The Track Update Thread, residing on processor P8, was the heaviest user of the Tracking Nodal Memory M4, as shown in Figure 3-19. When the Track Update Segment 19 began, M3's memory was decremented from its previous state in order to simulate the dropped tracks from the Object Track File located on M3. However, as new tracks were received, the database size increased to its previous level for the duration of the run.

Another test case used in analyzing the Tracking Nodal Memory utilization with 50 objects in the system was reducing the Memory Capacity threshold from eight megabytes to only 500,000 bytes. As seen in the partial textual listing of the Resource Use Profile shown in Figure 3-20, this limit was exceeded by 60% at the start of the run. The maximum percent usage at 500,000 bytes for M5 was 160%. In other words, this Track Model nodal memory requires at least 800,000 bytes of storage capacity.

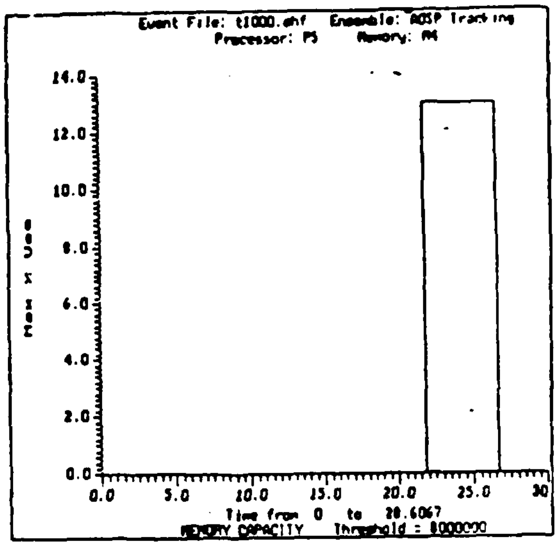
N = 1000 Objects In the System

Processor P3



Increments Object State Data File

Processor P5

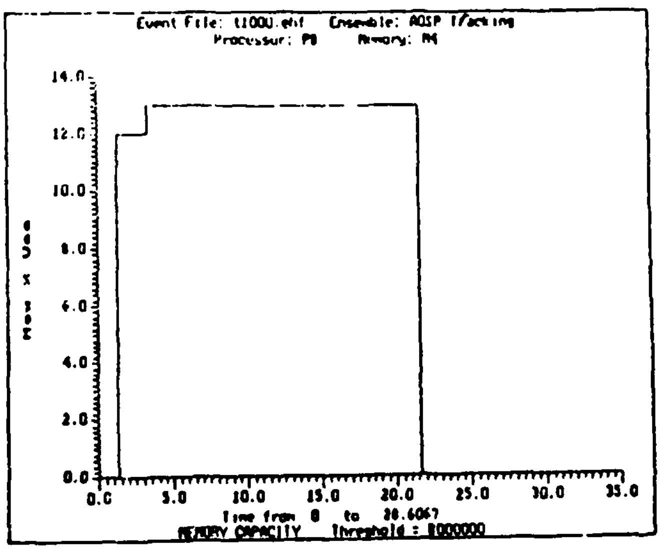


Increments Object Track File

Figure 3-18 Tracking Nodal Memory Capacity Part 1

N = 1000 Objects In the System

Processor P8



Loads instruction set & steady state data sizes
for Object Track File & Object State Data File
Decrements Object Track File

Figure 3-19 Tracking Nodal Memory Capacity Part 2

```
****************************************
*****      RESOURCE USE LISTING     *****
****************************************

Resource Use profile of    MEMORY CAPACITY    for    M5

Interval:  0   to   2

Memory Size = 500000

Event Information from file :  M500K.ehf
     Processor Ensemble : AOSP Tracking
        Author: TBE

     MAXIMUM USE OF RESOURCE IN TIME INTERVALS

 interval          start time          end time              %used
        0                   0          0.0124045               160
        1           0.0124045          0.0248089               160
        2           0.0248089          0.0372134               160
        3           0.0372134          0.0496179               160
        4           0.0496179          0.0620223               160
        5           0.0620223          0.0744268
```

Figure 3-20 Resource Use Profile for M5 with 50 Objects in the System

The next resource evaluated was the Bus Bandwidth. Although there are many messages passed between the AOA software functions, the extremely fast bus bandwidth on the AOSP hardware architecture, 12 megabytes per second, almost made the data traffic seem nonexistent. The only busses with significant utilization for the 1,000 object case were busses B1, B2, and B3. The graphs of the maximum percentage of bus bandwidth use for these busses are shown in Figure 3-21. Bus traffic for the data links between the Object Screening Node and the Tracking Nodes (B1 & B2) consisted of Uncorrelated Sighting Messages, Missed Sighting Messages, Track Update Messages, and Predicted Window File Data. Traffic from the Sorting Node to Screening Node was Object Sighting Reports. The percentage of bus bandwidth use for all the other busses (B4, B5, B6, & B7) was approximately zero over the entire simulation run.

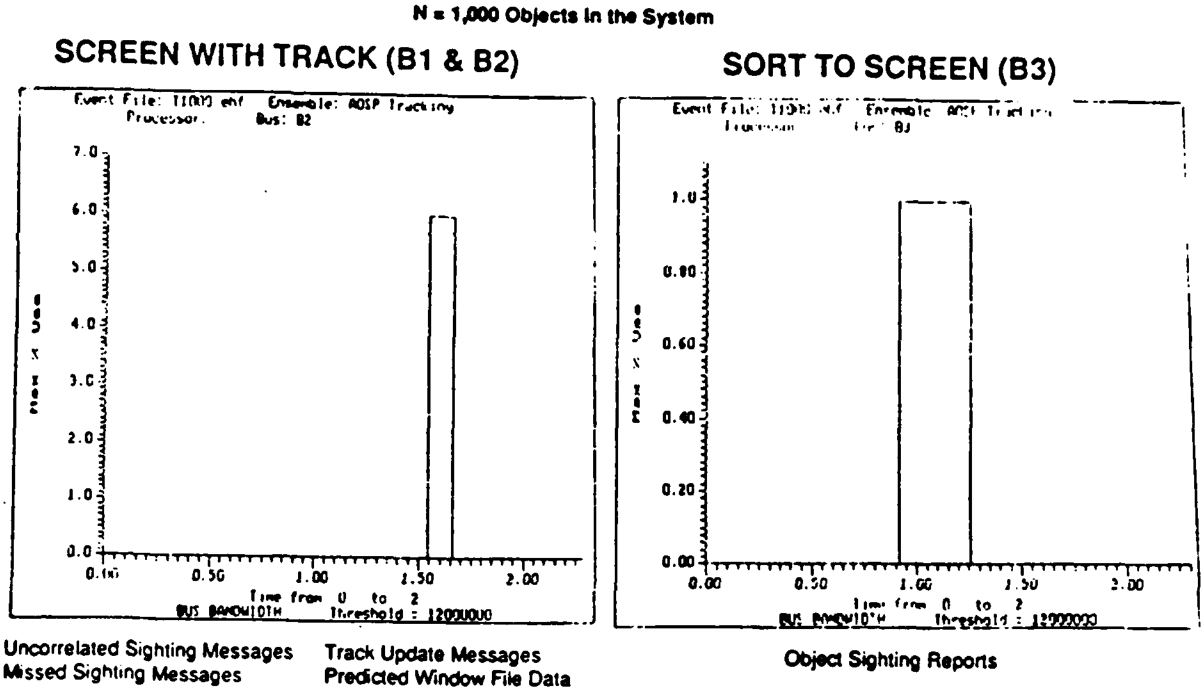
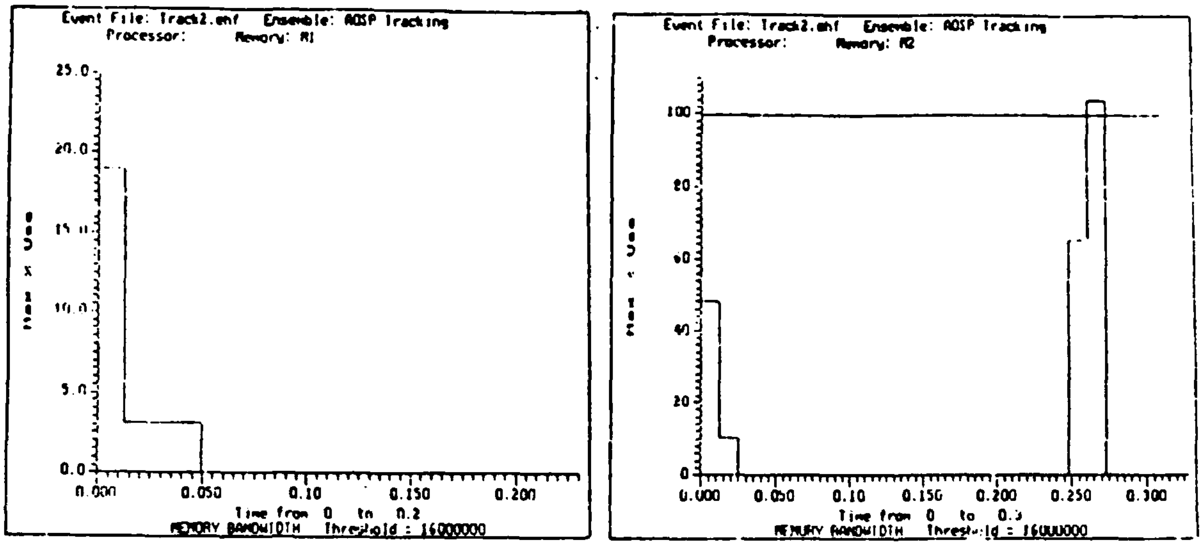


Figure 3-21 Track Model Bus Bandwidth Analysis

The Track Model Nodal Memory Bandwidth Analysis proved to be the most interesting of the resource analyses, since the percent of the memory bandwidth for each nodal memory was significant. Shown in Figure 3-22, the Object Sorting Nodal Memory's bandwidth usage reached almost 20% for 50 objects, while the Object Screening Nodal Memory's maximum memory bandwidth was exceeded for the same number of objects. Shown in Figure 3-23, the Navigation, Input/Output, and Control Nodal Memory only reached 10% of its maximum threshold with 50 objects, while the Tracking Nodal Memories reached almost 20% of their thresholds. The NIOC nodal accesses were primarily for the Navigational History File. The Tracking Nodal Memory accesses were needed for the management of both the Object State Data File and the Object Track File.

N = 50 Objects In the System

OBJECT SORTING NODAL MEMORY (M1) **OBJECT SCREENING NODAL MEMORY (M2)**

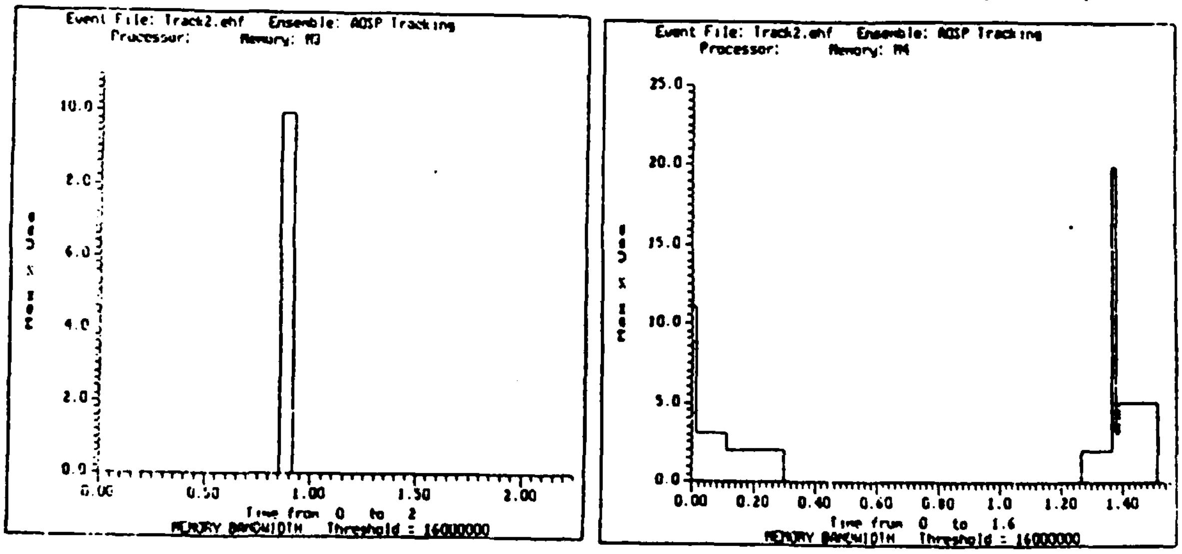


* Note that the scales are different for each graph

Figure 3-22 Track Model Nodal Memory Bandwidth Analysis Part 1

N = 50 Objects In the System

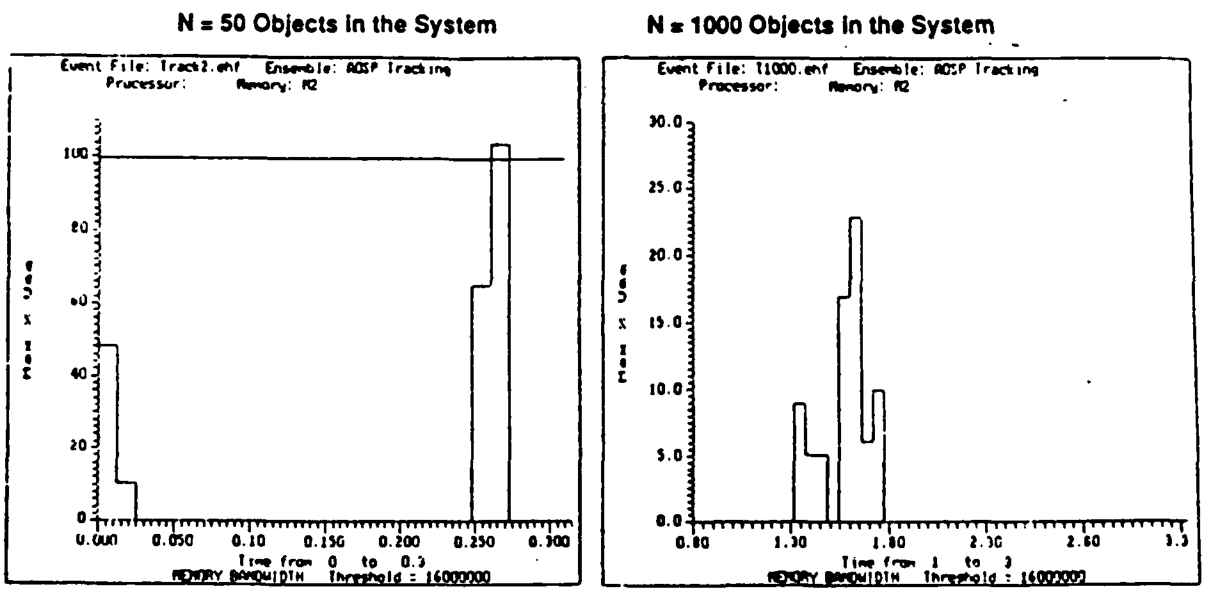
NIOC NODAL MEMORY (M3) **TRACKING NODAL MEMORY (M4 & M5)**



* Note that the scales are different for each graph

Figure 3-23 Track Model Nodal Memory Bandwidth Analysis Part 2

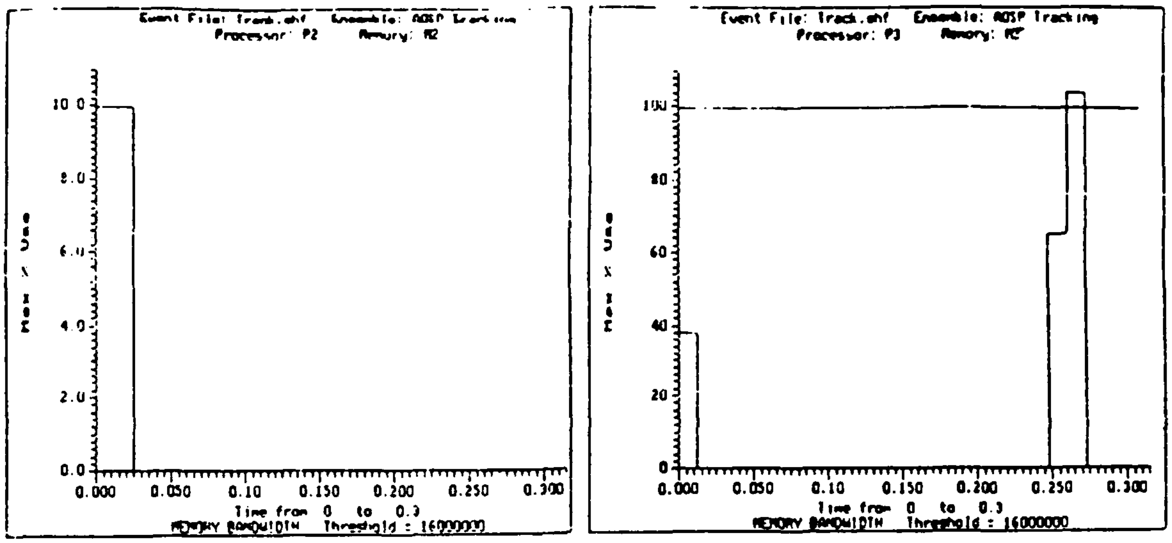
As shown in Figure 3-24, with only 50 objects in the system, the Object Screening Nodal Memory's bandwidth threshold of 16 megabytes/second was exceeded. However, with 1,000 objects in the system, the maximum percent usage was under 25%. To gain further insight into the possible cause of the Object Screening Nodal Memory bandwidth threshold being exceeded, the two processors that had access to this memory were evaluated. As shown in the graph on the left of Figure 3-25, the segments running on P2 did not exceed the threshold. The diagram on the right shows that a segment running on P3 exceeded this threshold. Comparing the specific time of exceedance with the event list showed that Predicted Window File Sort Segment 10 was the segment running when the limit was passed. Since the runtime for this segment was extremely small, 0.00045 seconds with 50 objects, the segment did not have enough time to access needed data before the runtime was completed. In the worst case, this would mean that the segment's runtime is only off by a few thousandths of a second, causing no major delays. With more objects in the system, as seen in Figure 3-24, the memory bandwidth threshold would not be exceeded.



* Note that the scales are different for each graph

Figure 3-24 Object Screening Nodal Memory Bandwidth Analysis Part 1

Processor P2 | Processor P3



Decrements Predicted Window File

Increments, Decrements, & Reads
Object Irradiance File
Increments Failed Track List File
Increments Predicted Window File

* Note that the scales are different for each graph

Figure 3-25 Object Screening Nodal Memory Bandwidth Analysis Part 2

3.7 TRACK MODEL ANALYSIS CONCLUSIONS AND RECOMMENDATIONS

Upon evaluating the test case results, the resource analyses of memory capacity and bus bandwidth showed no constraint problems when the AOA software was mapped onto the AOSP hardware with 1,000 objects in the system. The memory bandwidth constraint was only exceeded once at 50 objects, and its effect on the system would be negligible. Therefore, the AOA software can be feasibly mapped onto the AOSP hardware architecture. In fact, the AOA software would seem to map easily onto an AOSP hardware architecture with little timing problems.

Since the AOA software did not stress the limits of the AOSP architecture, an even more sophisticated software tracking system, like a Space-Based Surveillance Tracking System, could even be a possible candidate for this architecture. This would be an interesting follow-on analysis study.

3.8 PERM ANALYSIS CONCLUSIONS AND RECOMMENDATIONS

PERM is relatively user-friendly. Data entry, though somewhat tedious, is self-explanatory with the help of pop-up menus and scroll options. However, some input features could be "fine-tuned" for more efficient use.

PERM provides relatively "quick" response. Computational runs are relatively fast, and graphs can be produced in seconds by reducing data with the selected options. Organization can also speed data entry, thus increasing turn-around rate. System design charts, like those shown in Figures 3-6 and 3-8, were found to be very useful in helping to input data into PERM.

Like all simulation tools, PERM has limitations. One constraint, its memory, could be relaxed by employing more efficient software methods or by porting the software to a workstation.

PERM could be made much more flexible with the addition of a few options. For instance, some options to increase statistical accuracy could be included in PERM. An easy feature to implement would be to keep track of the percent of time that a particular processor is utilized over the entire run duration. These runtimes for the segments are already in the PERM databases, so the additions to the software would be minimal, yet the analyst would be provided some needed statistics for systems analysis studies. Also, adding an option to execute Monte-Carlo runs would automate the tedious procedure of regenerating random numbers and producing multiple runs for statistical accuracy.

Input specifications could also be made more independent. For example, by including as an option the ability to use the number of instructions executed per segment in the runtime equations, instead of time, the model software would be more independent of the hardware inputs. Also, by allowing the input variable to be changed from one segment to another may complicate the final analysis, but would free the user from dependance on a single variable for all his equations. Allowing multiple input variables would even further simplify the derivation of these equations.

PERM's fidelity could be increased by modifying the software to implement certain features. One of these features is resource utilization calculations. The current method PERM uses to calculate resource utilization is based on runtimes of segments. This may not always be representative of every system. For example, the analyst may need to simulate a segment passing data to another function, and the time for this data transfer is at the end of the segment run. However, the bandwidth usage is calculated in PERM by dividing the bytes accessed by the maximum number of bytes allowed during the segment's entire runtime. Therefore, the data traffic distribution is not realistically modeled. One way PERM could more accurately model the resource utilization distribution would be to allow the analyst to input transfer equations for each start time of the resource utilization (based on a delta of the runtime) and duration times for each resource utilization.

A suggested enhancement described in detail in section 3.6 of this report is to change the way PERM models memory usage. This could be accomplished by using delta values for each segment's memory usage transfer equations that would change the total memory used on a node. In this way, databases would be more realistically modeled.

A simple modification to increase fidelity would be to increase the arithmetic precision of all the PERM calculations. By having more precise results, the analyst would be able to better monitor resource activity.

PERM is meant to be used as a feasibility tool. When performing detailed analysis, PERM should be used with, not instead of, a higher fidelity simulation.

APPENDIX A

LISTING OF PERM TEST CASE

APPENDIX A

A.0 INTRODUCTION

This Appendix provides a complete listing of the PERM Test Case constructed to support testing and demonstration of the system. This Test Case is referred to repeatedly throughout Section 1 of the Final Report. The material is a verbatim reproduction of the listings produced the the PERM Print and Log commands in IPERM.

The first section presents the Processor Ensemble definition. The second section presents the three Task Classes constructed for the example. The third presents the full validate System Load.

A.1 Processor Ensemble Definition

Processor Ensemble Name: TEST CASE
Author: SPARTA Creation Date: 07/24/89
Validated = TRUE
Validation Date: 07/26/89 Validation Time: 14:26:47.52
*Processor Class List:
**Processor Class Name: DMA

*****Accessible Processors Variables List:**

Variable Name:	Class Restriction:
v_local_cpu	CPU
v_other_dma	DMA

*****Accessible Memories Variables List:**

Variable Name:	Class Restriction:
v_local_mem	LOCAL MEMORY
v_global_mem	GLOBAL MEMORY

***Accessible Busses Variables List:

Variable Name:	Class Restriction
v_io_bus	I/O BUS

***Processor Class Instantiation List:

****Processor Name: dma_2
Processor Class: DMA

*****Accessible Processors Variables Assignment List:

Variable Name:	=> Assigned Value:
v_local_cpu	=> cpu_2
v_other_dma	=> dma_1

*****Accessible Memories Variables Assignment List:

Variable Name:	=> Assigned Value:
v_local_mem	=> mem_2
v_global_mem	=> glob_mem

*****Accessible Busses Variables Assignment List:

Variable Name:	=> Assigned Value:
v_io_bus	=> io_bus

****Processor Name: dma_1
Processor Class: DMA

*****Accessible Processors Variables Assignment List:

Variable Name:	=> Assigned Value:
v_local_cpu	=> cpu_1
v_other_dma	=> dma_2

*****Accessible Memories Variables Assignment List:

Variable Name:	**=> Assigned Value:**
v_local_mem	=> mem_1
v_global_mem	=> glob_mem

*****Accessible Busses Variables Assignment List:**

Variable Name:	**=> Assigned Value:**
v_io_bus	=> io_bus

**Processor Class Name: CPU

*****Accessible Processors Variables List:**

Variable Name:	**Class Restriction:**
v_local_dma	DMA

*****Accessible Memories Variables List:**

Variable Name:	**Class Restriction:**
v_local_mem	LOCAL MEMORY

*****Accessible Busses Variables List:**

Variable Name:	**Class Restriction**

— List Empty —

***Processor Class Instantiation List:

****Processor Name: cpu_2

Processor Class: CPU

*******Accessible Processors Variables Assignment List:**

Variable Name:	**=> Assigned Value:**
v_local_dma	=> dma_2

*******Accessible Memories Variables Assignment List:**

Variable Name:	**=> Assigned Value:**

v_local_mem => mem_2

*******Accessible Busses Variables Assignment List:**

Variable Name:	**=> Assigned Value:**
— List Empty —	

****Processor Name: cpu_1
Processor Class: CPU

*******Accessible Processors Variables Assignment List:**

Variable Name:	**=> Assigned Value:**
v_local_dma	=> dma_1

*******Accessible Memories Variables Assignment List:**

Variable Name:	**=> Assigned Value:**
v_local_mem	=> mem_1

*******Accessible Busses Variables Assignment List:**

Variable Name:	**=> Assigned Value:**
— List Empty —	

*Memory Class List:
**Memory Class Name: GLOBAL MEMORY
Size: 20000000 Bytes
I/O Band Width: 10000000 Bytes/Second

*****Client Processors Variables List:**

Variable Name:	**Class Restriction:**
v_first_dma	DMA
v_second_dma	DMA

***Memory Class Instantiation List:

****Memory Name: glob_mem
Memory Class: GLOBAL MEMORY

*****Client Processors Variables Assignment List:

Variable Name:	=> Assigned Value:
v_first_dma	=> dma_1
v_second_dma	=> dma_2

**Memory Class Name: LOCAL MEMORY
Size: 4000000 Bytes
I/O Band Width: 30000000 Bytes/Second

***Client Processors Variables List:

Variable Name:	Class Restriction:
v_local_cpu	CPU
v_local_dma	DMA

***Memory Class Instantiation List:
****Memory Name: mem_2
Memory Class: LOCAL MEMORY

*****Client Processors Variables Assignment List:

Variable Name:	=> Assigned Value:
v_local_cpu	=> cpu_2
v_local_dma	=> dma_2

****Memory Name: mem_1
Memory Class: LOCAL MEMORY

*****Client Processors Variables Assignment List:

Variable Name:	=> Assigned Value:
v_local_cpu	=> cpu_1

v_local_dma => dma_1

*Bus Class List:

**Bus Class Name: I/O BUS
Effective Band Width: 10000000 Bytes/Second

***Client Processors Variables List:

Variable Name:	Class Restriction:
v_first_dma	DMA
v_second_dma	DMA

***Bus Class Instantiation List:

****Bus Name: io_bus
Bus Class: I/O BUS

*****Client Processors Variables Assignment List:

Variable Name:	=> Assigned Value:
v_first_dma	=> dma_1
v_second_dma	=> dma_2

A.2 Task Class Definitions

This section presents the three Task Class definitions constructed for the Test Case example. This includes the full Segment definitions (including all transfer functions) and Thread definitions.

A.2.1 Tracking

Task Class Name: TRACKING

Processor Ensemble Name: TEST CASE

Author: SPARTA

Creation Date: 07/24/89 Validated: FALSE

Last Validation Date: 07/26/89 Last Validation Time: 14:27:26.68

****Input Start Time Dependency Variables List:**

Variable Name:	**Task Class Restriction:**
v_prior_ephemeris_gen	EPHEMERIS GENERATION

****Output Start Time Dependency Variables List:**

Variable Name:	**Task Class Restriction:**
v_subsequent_ephem_gen	EPHEMERIS GENERATION

****Segment Class Table**

*****Segment Class Name: COMPUTE PHASE FIVE**

Target Processor Class: CPU

Number of Instantiations: 2

Segment Class Type: Application Code

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 5.000000E-01	Q1 = 3.000000E-05	L1 = 0.000000E+00
	Q2 = 5.000000E-04	L2 = 6.000000E-05
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 5.000000E-01	Q1 = 7.000000E+04	L1 = 0.000000E+00
	Q2 = 3.200000E+02	L2 = 1.000000E+01
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 5.000000E-01	Q1 = 1.800000E+02	L1 = 0.000000E+00
	Q2 = 6.500000E+03	L2 = 4.200000E+02
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

*****Segment Class Name: CLEAN UP**

Target Processor Class: CPU

Number of Instantiations: 2

Segment Class Type: Operating System

****Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M2) + (L1 +(L2 * M))LOG2(M)**

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 3.000000E-04	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 8.000000E+03	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 4.500000E+03	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

.

*****Segment Class Name: STORE BACK TO GLOBAL MEMORY**

Target Processor Class: DMA

Number of Instantiations: 2

Segment Class Type: Application Code

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 5.000000E-01	Q1 = 1.000000E-03	L1 = 0.000000E+00
	Q2 = 5.000000E-05	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 5.000000E-01	Q1 = 5.000000E+04	L1 = 0.000000E+00
	Q2 = 2.000000E+02	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

.

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 5.000000E-01 Q1 = 0.000000E+00 L1 = 0.000000E+00

Q2 = 2.000000E+02 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 5.000000E-01 Q1 = 5.000000E+03 L1 = 0.000000E+00

Q2 = 2.000000E+02 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 5.000000E-01 Q1 = 5.000000E+03 L1 = 0.000000E+00

Q2 = 2.000000E+02 L2 = 0.000000E+00

Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: v_io_bus

Bus Variable Class Restriction: I/O BUS

R = 5.000000E-01 Q1 = 5.000000E+03 L1 = 0.000000E+00

Q2 = 2.000000E+02 L2 = 0.000000E+00

Q3 = 0.000000E+00

*****Segment Class Name: COMPUTE PHASE SIX**

Target Processor Class: CPU

Number of Instantiations: 2

Segment Class Type: Application Code

****Transfer Functions List:

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 5.000000E-01	Q1 = 8.000000E-03	L1 = 0.000000E+00
	Q2 = 7.000000E-05	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 5.000000E-01	Q1 = 5.000000E+04	L1 = 0.000000E+00
	Q2 = 3.000000E+02	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 5.000000E-01	Q1 = 8.000000E+04	L1 = 0.000000E+00
	Q2 = 9.100000E+02	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

***Segment Class Name: COMPUTE PHASE FOUR

Target Processor Class: CPU

Number of Instantiations: 2

Segment Class Type: Application Code

****Transfer Functions List:

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 5.000000E-01	Q1 = 1.800000E-04	L1 = 0.000000E+00
	Q2 = 7.200000E-05	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 5.000000E-01	Q1 = 5.000000E+04	L1 = 0.000000E+00
	Q2 = 3.000000E+02	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 5.000000E-01	Q1 = 2.200000E+03	L1 = 0.000000E+00
	Q2 = 6.600000E+02	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

*****Segment Class Name: RECEIVE DATA**

Target Processor Class: DMA

Number of Instantiations: 2

Segment Class Type: Application Code

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.500000E-01	Q1 = 2.000000E-06	L1 = 0.000000E+00
	Q2 = 1.470000E-04	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.500000E-01	Q1 = 5.000000E+04	L1 = 0.000000E+00
	Q2 = 9.000000E+02	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.500000E-01 Q1 = 0.000000E+00 L1 = 0.000000E+00

Q2 = 2.200000E+02 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: v_io_bus

Bus Variable Class Restriction: I/O BUS

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

*****Segment Class Name: SEND DATA**

Target Processor Class: DMA

Number of Instantiations: 2

Segment Class Type: Application Code

****Transfer Functions List:

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.500000E-01	Q1 = 2.000000E-06	L1 = 0.000000E+00
	Q2 = 1.470000E-05	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.500000E-01	Q1 = 5.000000E+04	L1 = 0.000000E+00
	Q2 = 6.700000E+02	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.500000E-01 Q1 = 0.000000E+00 L1 = 0.000000E+00

Q2 = 2.200000E+02 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: v_io_bus

Bus Variable Class Restriction: I/O BUS

R = 1.500000E-01 Q1 = 0.000000E+00 L1 = 0.000000E+00

Q2 = 2.200000E+02 L2 = 0.000000E+00

Q3 = 0.000000E+00

*****Segment Class Name: SEND DATA LAG**

Target Processor Class: DMA

Number of Instantiations: 2

Segment Class Type: Application Code

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 7.000000E-05	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 5.000000E-01	Q1 = 5.000000E+04	L1 = 0.000000E+00
	Q2 = 2.000000E+02	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 1.300000E+03	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: v_io_bus

Bus Variable Class Restriction: I/O BUS

R = 1.000000E+00 Q1 = 1.200000E+03 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

*****Segment Class Name: COMPUTE PHASE THREE**

Target Processor Class: CPU

Number of Instantiations: 2

Segment Class Type: Application Code

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 5.000000E-01 Q1 = 5.000000E-05 L1 = 0.000000E+00

Q2 = 1.500000E-05 L2 = 0.000000E+00

Q3 = 2.100000E-06

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 5.000000E-01	Q1 = 5.000000E+04	L1 = 0.000000E+00
	Q2 = 2.700000E+02	L2 = 0.000000E+00
	Q3 = 3.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 5.000000E-01	Q1 = 8.500000E+02	L1 = 0.000000E+00
	Q2 = 1.050000E+02	L2 = 0.000000E+00
	Q3 = 5.400000E+01	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

*****Segment Class Name: STATUS**

Target Processor Class: CPU

Number of Instantiations: 2

Segment Class Type: Operating System

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 3.700000E-03	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 5.000000E-01	Q1 = 5.000000E+04	L1 = 0.000000E+00
	Q2 = 2.000000E+02	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 7.000000E+04	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

*****Segment Class Name: COMPUTE PHASE TWO**

Target Processor Class: CPU

Number of Instantiations: 2

Segment Class Type: Application Code

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M2) + (L1 +(L2 * M))LOG2(M)**

Run Time Transfer Function Coeficients:

R = 5.000000E-01	Q1 = 1.500000E-03	L1 = 0.000000E+00
	Q2 = 9.000000E-05	L2 = 2.200000E-05
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 5.000000E-01	Q1 = 5.000000E+04	L1 = 0.000000E+00
	Q2 = 2.500000E+02	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 5.000000E-01	Q1 = 2.600000E+04	L1 = 0.000000E+00
	Q2 = 7.600000E+02	L2 = 4.100000E+02
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

***Segment Class Name: COMPUTE PHASE ONE

Target Processor Class: CPU

Number of Instantiations: 2

Segment Class Type: Application Code

****Transfer Functions List:

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))*LOG2(M)

Run Time Transfer Function Coeficients:

R = 5.000000E-01	Q1 = 7.000000E-04	L1 = 0.000000E+00
	Q2 = 4.600000E-04	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 5.000000E-01	Q1 = 5.000000E+04	L1 = 0.000000E+00
	Q2 = 2.000000E+02	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Va[illegible]e Class Restriction: LOCAL MEMORY

R = 5.000000E-01	Q1 = 7.500000E+03	L1 = 0.000000E+00
	Q2 = 1.320000E+04	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

*****Segment Class Name: GET DATA**

Target Processor Class: DMA

Number of Instantiations: 2

Segment Class Type: Application Code

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 5.000000E-01	Q1 = 7.000000E-04	L1 = 0.000000E+00
	Q2 = 5.000000E-05	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 5.000000E-01	**Q1 = 0.000000E+00**	**L1 = 0.000000E+00**
	Q2 = 2.000000E+02	**L2 = 0.000000E+00**
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 5.000000E-01	**Q1 = 5.000000E+03**	**L1 = 0.000000E+00**
	Q2 = 2.000000E+02	**L2 = 0.000000E+00**
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 5.000000E-01	**Q1 = 5.000000E+03**	**L1 = 0.000000E+00**
	Q2 = 2.000000E+02	**L2 = 0.000000E+00**
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: v_io_bus

Bus Variable Class Restriction: I/O BUS

R = 5.000000E-01	**Q1 = 5.000000E+03**	**L1 = 0.000000E+00**
	Q2 = 2.000000E+02	**L2 = 0.000000E+00**
	Q3 = 0.000000E+00	

*****Segment Class Name: SET UP TABLE**

Target Processor Class: CPU

Number of Instantiations: 2

Segment Class Type: Application Code

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

$F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M^{**}2) + (L1 + (L2 * M))LOG2(M)$

Run Time Transfer Function Coeficients:

R = 5.000000E-01	Q1 = 4.500000E-03	L1 = 0.000000E+00
	Q2 = 2.000000E-05	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 5.000000E-01	Q1 = 5.000000E+04	L1 = 0.000000E+00
	Q2 = 2.000000E+02	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 5.000000E-01	Q1 = 3.800000E+04	L1 = 0.000000E+00
	Q2 = 1.840000E+02	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

*****Segment Class Name: FORK PROCESS**

Target Processor Class: CPU

Number of Instantiations: 2

Segment Class Type: Operating System

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 1.000000E-03	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 8.000000E+03	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 1.000000E+04	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

*****Segment Class Name: LOAD INSTRUCTIONS**

Target Processor Class: DMA

Number of Instantiations: 2

Segment Class Type: Operating System

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 2.000000E-02	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00 Q1 = 4.000000E+04 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00 Q1 = 3.000000E+04 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00 Q1 = 3.200000E+04 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00 Q1 = 3.200000E+04 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: v_io_bus

Bus Variable Class Restriction: I/O BUS

R = 1.000000E+00 Q1 = 3.300000E+04 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

***Segment Class Name: SYSTEM INITIALIZATION

Target Processor Class: CPU

Number of Instantiations: 2

Segment Class Type: Operating System

****Transfer Functions List:

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 3.000000E-03	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 8.000000E+03	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00 Q1 = 5.000000E+04 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

*****Segment Class Name: DMA JOIN**

Target Processor Class: DMA

Number of Instantiations: 9

Segment Class Type: Join

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: v_io_bus

Bus Variable Class Restriction: I/O BUS

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

*****Segment Class Name: CPU JOIN**

Target Processor Class: CPU

Number of Instantiations: 6

Segment Class Type: Join

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

****Thread Table**

*****Thread Name: Third Tracking Thread: CPU**

Target Processor: cpu_2

Target Processor Class: CPU

******Segment List:**

Segment Name: Thread 3 - Segment 1

Segment Class: SYSTEM INITIALIZATION

Segment Type: Operating System

Segment Name: Thread 3 - Segment 2

Segment Class: CPU JOIN

Segment Type: Join

Predecessor Segment: Thread 4 - Segment 2

Predecessor Thread: Fourth Tracking Thread: DMA

Segment Name: Thread 3 - Segment 3

Segment Class: FORK PROCESS

Segment Type: Operating System

Segment Name: Thread 3 - Segment 4

Segment Class: SET UP TABLE

Segment Type: Application Code

Segment Name: Thread 3 - Segment 5

Segment Class: CPU JOIN

Segment Type: Join

Predecessor Segment: Thread 4 - Segment 4

Predecessor Thread: Fourth Tracking Thread: DMA

Segment Name: Thread 3 - Segment 6

Segment Class: COMPUTE PHASE ONE

Segment Type: Application Code

Segment Name: Thread 3 - Segment 7

Segment Class: COMPUTE PHASE TWO

Segment Type: Application Code

Segment Name: Thread 3 - Segment 8

Segment Class: STATUS

Segment Type: Operating System

Segment Name: Thread 3 - Segment 9

Segment Class: COMPUTE PHASE THREE

Segment Type: Application Code

Segment Name: Thread 3 - Segment 10

Segment Class: CPU JOIN

Segment Type: Join

Predecessor Segment: Thread 4 - Segment 6

Predecessor Thread: Fourth Tracking Thread: DMA

Segment Name: Thread 3 - Segment 11

Segment Class: COMPUTE PHASE FOUR

Segment Type: Application Code

Segment Name: Thread 3 - Segment 12

Segment Class: COMPUTE PHASE FIVE

Segment Type: Application Code

Segment Name: Thread 3 - Segment 13

Segment Class: COMPUTE PHASE SIX

Segment Type: Application Code

Segment Name: Thread 3 - Segment 14

Segment Class: CLEAN UP

Segment Type: Operating System

*****Thread Name: First Tracking Thread: CPU**

Target Processor: cpu_1

Target Processor Class: CPU

******Segment List:**

Segment Name: Thread 1 - Segment 1

Segment Class: SYSTEM INITIALIZATION

Segment Type: Operating System

Segment Name: Thread 1 - Segment 2

Segment Class: CPU JOIN

Segment Type: Join

Predecessor Segment: Thread 2 - Segment 2
Predecessor Thread: Second Tracking Thread: DMA

Segment Name: Thread 1 - Segment 3
Segment Class: FORK PROCESS
Segment Type: Operating System

Segment Name: Thread 1 - Segment 4
Segment Class: SET UP TABLE
Segment Type: Application Code

Segment Name: Thread 1 - Segment 5
Segment Class: CPU JOIN
Segment Type: Join
Predecessor Segment: Thread 2 - Segment 4
Predecessor Thread: Second Tracking Thread: DMA

Segment Name: Thread 1 - Segment 6
Segment Class: COMPUTE PHASE ONE
Segment Type: Application Code

Segment Name: Thread 1 - Segment 7
Segment Class: COMPUTE PHASE TWO
Segment Type: Application Code

Segment Name: Thread 1 - Segment 8
Segment Class: STATUS
Segment Type: Operating System

Segment Name: Thread 1 - Segment 9

Segment Class: COMPUTE PHASE THREE

Segment Type: Application Code

Segment Name: Thread 1 - Segment 10

Segment Class: CPU JOIN

Segment Type: Join

Predecessor Segment: Thread 2 - Segment 9

Predecessor Thread: Second Tracking Thread: DMA

Segment Name: Thread 1 - Segment 11

Segment Class: COMPUTE PHASE FOUR

Segment Type: Application Code

Segment Name: Thread 1 - Segment 12

Segment Class: COMPUTE PHASE FIVE

Segment Type: Application Code

Segment Name: Thread 1 - Segment 13

Segment Class: COMPUTE PHASE SIX

Segment Type: Application Code

Segment Name: Thread 1 - Segment 14

Segment Class: CLEAN UP

Segment Type: Operating System

*****Thread Name: Second Tracking Thread: DMA**

Target Processor: dma_1

Target Processor Class: DMA

****Segment List:

Segment Name: Thread 2 - Segment 1

Segment Class: DMA JOIN

Segment Type: Join

Predecessor Segment: Thread 1 - Segment 1

Predecessor Thread: First Tracking Thread: CPU

Segment Name: Thread 2 - Segment 2

Segment Class: LOAD INSTRUCTIONS

Segment Type: Operating System

Segment Name: Thread 2 - Segment 3

Segment Class: DMA JOIN

Segment Type: Join

Predecessor Segment: Thread 1 - Segment 3

Predecessor Thread: First Tracking Thread: CPU

Segment Name: Thread 2 - Segment 4

Segment Class: GET DATA

Segment Type: Application Code

Segment Name: Thread 2 - Segment 5

Segment Class: DMA JOIN

Segment Type: Join

Predecessor Segment: Thread 1 - Segment 9

Predecessor Thread: First Tracking Thread: CPU

Segment Name: Thread 2 - Segment 6
Segment Class: SEND DATA LAG
Segment Type: Application Code

Segment Name: Thread 2 - Segment 7
Segment Class: SEND DATA
Segment Type: Application Code

Segment Name: Thread 2 - Segment 8
Segment Class: DMA JOIN
Segment Type: Join
Predecessor Segment: Thread 4 - Segment 7
Predecessor Thread: Fourth Tracking Thread: DMA

Segment Name: Thread 2 - Segment 9
Segment Class: RECEIVE DATA
Segment Type: Application Code

Segment Name: Thread 2 - Segment 10
Segment Class: DMA JOIN
Segment Type: Join
Predecessor Segment: Thread 1 - Segment 13
Predecessor Thread: First Tracking Thread: CPU

Segment Name: Thread 2 - Segment 11
Segment Class: STORE BACK TO GLOBAL MEMORY
Segment Type: Application Code

*****Thread Name: Fourth Tracking Thread: DMA**

Target Processor: dma_2

Target Processor Class: DMA

******Segment List:**

Segment Name: Thread 4 - Segment 1

Segment Class: DMA JOIN

Segment Type: Join

Predecessor Segment: Thread 3 - Segment 1

Predecessor Thread: Third Tracking Thread: CPU

Segment Name: Thread 4 - Segment 2

Segment Class: LOAD INSTRUCTIONS

Segment Type: Operating System

Segment Name: Thread 4 - Segment 3

Segment Class: DMA JOIN

Segment Type: Join

Predecessor Segment: Thread 3 - Segment 4

Predecessor Thread: Third Tracking Thread: CPU

Segment Name: Thread 4 - Segment 4

Segment Class: GET DATA

Segment Type: Application Code

Segment Name: Thread 4 - Segment 5

Segment Class: DMA JOIN

Segment Type: Join

Predecessor Segment: Thread 2 - Segment 6

Predecessor Thread: Second Tracking Thread: DMA

Segment Name: Thread 4 - Segment 6

Segment Class: RECEIVE DATA

Segment Type: Application Code

Segment Name: Thread 4 - Segment 7

Segment Class: SEND DATA LAG

Segment Type: Application Code

Segment Name: Thread 4 - Segment 8

Segment Class: SEND DATA

Segment Type: Application Code

Segment Name: Thread 4 - Segment 9

Segment Class: DMA JOIN

Segment Type: Join

Predecessor Segment: Thread 3 - Segment 13

Predecessor Thread: Third Tracking Thread: CPU

Segment Name: Thread 4 - Segment 10

Segment Class: STORE BACK TO GLOBAL MEMORY

Segment Type: Application Code

A.2.2 Ephemeris Generation

Task Class Name: EPHEMERIS GENERATION

Processor Ensemble Name: TEST CASE

Author: SPARTA

Creation Date: 07/25/89   Validated: TRUE

Last Validation Date: 07/25/89   Last Validation Time: 14:08:37.36

****Input Start Time Dependency Variables List:**

Variable Name:	**Task Class Restriction:**
v_prior_tracking	TRACKING

****Output Start Time Dependency Variables List:**

Variable Name:	Task Class Restriction:
v_subsequent_tracking	TRACKING

****Segment Class Table**

*****Segment Class Name: PROCESS DATA**

Target Processor Class: CPU

Number of Instantiations: 1

Segment Class Type: Application Code

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 3.000000E-03 L1 = 0.000000E+00

Q2 = 9.200000E-04 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00 Q1 = 1.800000E+04 L1 = 0.000000E+00

Q2 = 8.000000E+01 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00 Q1 = 5.400000E+04 L1 = 0.000000E+00

Q2 = 1.530000E+04 L2 = 0.000000E+00

Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

*****Segment Class Name: CPU JOIN**

Target Processor Class: CPU

Number of Instantiations: 2

Segment Class Type: Join

****Transfer Functions List:

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

***Segment Class Name: DMA JOIN

Target Processor Class: DMA

Number of Instantiations: 3

Segment Class Type: Join

****Transfer Functions List:

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: v_io_bus

Bus Variable Class Restriction: I/O BUS

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

***Segment Class Name: SYSTEM INITIALIZATION

Target Processor Class: CPU

Number of Instantiations: 1

Segment Class Type: Operating System

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M2) + (L1 +(L2 * M))LOG2(M)**

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 3.000000E-03	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 8.000000E+03	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 5.000000E+04	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

*****Segment Class Name: LOAD INSTRUCTIONS**

Target Processor Class: DMA

Number of Instantiations: 1

Segment Class Type: Operating System

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M2) + (L1 +(L2 * M))LOG2(M)**

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 2.000000E-02	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 4.000000E+04	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00	Q1 = 3.000000E+04	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	**Q1 = 3.200000E+04**	**L1 = 0.000000E+00**
	Q2 = 0.000000E+00	**L2 = 0.000000E+00**
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00	**Q1 = 3.200000E+04**	**L1 = 0.000000E+00**
	Q2 = 0.000000E+00	**L2 = 0.000000E+00**
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: v_io_bus

Bus Variable Class Restriction: I/O BUS

R = 1.000000E+00	**Q1 = 3.300000E+04**	**L1 = 0.000000E+00**
	Q2 = 0.000000E+00	**L2 = 0.000000E+00**
	Q3 = 0.000000E+00	

*****Segment Class Name: FORK PROCESS**

Target Processor Class: CPU

Number of Instantiations: 1

Segment Class Type: Operating System

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 1.000000E-03	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 8.000000E+03	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 1.000000E+04	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

*****Segment Class Name: CLEAN UP**

Target Processor Class: CPU

Number of Instantiations: 1

Segment Class Type: Operating System

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M2) + (L1 +(L2 * M))LOG2(M)**

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 3.000000E-04	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 8.000000E+03	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 4.500000E+03	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

*****Segment Class Name: GET DATA**

Target Processor Class: DMA

Number of Instantiations: 1

Segment Class Type: Application Code

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 1.000000E-03	L1 = 0.000000E+00
	Q2 = 7.500000E-06	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 1.800000E+04	L1 = 0.000000E+00
	Q2 = 6.000000E+01	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00	Q1 = 1.000000E+04	L1 = 0.000000E+00
	Q2 = 6.000000E+01	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 3.000000E+02	L1 = 0.000000E+00
	Q2 = 6.000000E+01	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00	Q1 = 3.000000E+02	L1 = 0.000000E+00
	Q2 = 6.000000E+01	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: v_io_bus

Bus Variable Class Restriction: I/O BUS

R = 1.000000E+00	Q1 = 8.000000E+02	L1 = 0.000000E+00
	Q2 = 6.000000E+01	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

*****Segment Class Name: STORE DATA**

Target Processor Class: DMA

Number of Instantiations: 1

Segment Class Type: Application Code

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 1.000000E-03 L1 = 0.000000E+00

Q2 = 7.500000E-06 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00 Q1 = 1.000000E+04 L1 = 0.000000E+00

Q2 = 6.000000E+01 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00 Q1 = 1.000000E+04 L1 = 0.000000E+00

Q2 = 6.000000E+01 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00 Q1 = 3.000000E+02 L1 = 0.000000E+00

Q2 = 6.000000E+01 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00 Q1 = 3.000000E+02 L1 = 0.000000E+00

Q2 = 6.000000E+01 L2 = 0.000000E+00

Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: v_io_bus

Bus Variable Class Restriction: I/O BUS

R = 1.000000E+00 Q1 = 8.000000E+02 L1 = 0.000000E+00

Q2 = 6.000000E+01 L2 = 0.000000E+00

Q3 = 0.000000E+00

****Thread Table**

*****Thread Name: Ephem_Gen First Thread: CPU**

Target Processor: cpu_1

Target Processor Class: CPU

******Segment List:**

Segment Name: Thread 1 - Segment 1

Segment Class: SYSTEM INITIALIZATION

Segment Type: Operating System

Segment Name: Thread 1 - Segment 2

Segment Class: CPU JOIN

Segment Type: Join

Predecessor Segment: Thread 2 - Segment 2

Predecessor Thread: Ephem_Gen Second Thread: DMA

Segment Name: Thread 1 - Segment 3

Segment Class: FORK PROCESS

Segment Type: Operating System

Segment Name: Thread 1 - Segment 4

Segment Class: CPU JOIN

Segment Type: Join

Predecessor Segment: Thread 2 - Segment 4

Predecessor Thread: Ephem_Gen Second Thread: DMA

Segment Name: Thread 1 - Segment 5

Segment Class: PROCESS DATA

Segment Type: Application Code

Segment Name: Thread 1 - Segment 6

Segment Class: CLEAN UP

Segment Type: Operating System

*****Thread Name: Ephem_Gen Second Thread: DMA**

Target Processor: dma_1

Target Processor Class: DMA

******Segment List:**

Segment Name: Thread 2 - Segment 1

Segment Class: DMA JOIN

Segment Type: Join

Predecessor Segment: Thread 1 - Segment 1

Predecessor Thread: Ephem_Gen First Thread: CPU

Segment Name: Thread 2 - Segment 2

Segment Class: LOAD INSTRUCTIONS

Segment Type: Operating System

Segment Name: Thread 2 - Segment 3

Segment Class: DMA JOIN

Segment Type: Join

Predecessor Segment: Thread 1 - Segment 3

Predecessor Thread: Ephem_Gen First Thread: CPU

Segment Name: Thread 2 - Segment 4

Segment Class: GET DATA

Segment Type: Application Code

Segment Name: Thread 2 - Segment 5

Segment Class: DMA JOIN

Segment Type: Join

Predecessor Segment: Thread 1 - Segment 5

Predecessor Thread: Ephem_Gen First Thread: CPU

Segment Name: Thread 2 - Segment 6

Segment Class: STORE DATA

Segment Type: Application Code

A.2.3 Sort and Search

Task Class Name: BIG SORT/SEARCH

Processor Ensemble Name: TEST CASE

Author: SPARTA

Creation Date: 07/25/89 Validated: TRUE

Last Validation Date: 07/26/89 Last Validation Time: 08:59:15.89

****Input Start Time Dependency Variables List:**

Variable Name: Task Class Restriction:

— List Empty —

****Output Start Time Dependency Variables List:**

Variable Name: Task Class Restriction:

— List Empty —

****Segment Class Table**

*****Segment Class Name: SORT DATA**

Target Processor Class: CPU

Number of Instantiations: 1

Segment Class Type: Application Code

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 5.120000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 1.020000E+06	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 7.420000E+07	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

*****Segment Class Name: PROCESS DATA**

Target Processor Class: CPU

Number of Instantiations: 4

Segment Class Type: Application Code

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 1.800000E-01	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 3.020000E+06	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 3.300000E+06	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

*****Segment Class Name: CPU JOIN**

Target Processor Class: CPU

Number of Instantiations: 5

Segment Class Type: Join

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M2) + (L1 +(L2 * M))LOG2(M)**

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

***Segment Class Name: DMA JOIN

Target Processor Class: DMA

Number of Instantiations: 6

Segment Class Type: Join

****Transfer Functions List:

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: v_io_bus

Bus Variable Class Restriction: I/O BUS

R = 1.000000E+00	Q1 = 0.000000E+00	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

*****Segment Class Name: SYSTEM INITIALIZATION**

Target Processor Class: CPU

Number of Instantiations: 1

Segment Class Type: Operating System

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 3.000000E-03	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 8.000000E+03	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 5.000000E+04	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

*****Segment Class Name: LOAD INSTRUCTIONS**

Target Processor Class: DMA

Number of Instantiations: 1

Segment Class Type: Operating System

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 2.000000E-02	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 4.000000E+04	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00	Q1 = 3.000000E+04	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 3.200000E+04	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00	Q1 = 3.200000E+04	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: v_io_bus

Bus Variable Class Restriction: I/O BUS

R = 1.000000E+00	Q1 = 3.300000E+04	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

***Segment Class Name: FORK PROCESS**

Target Processor Class: CPU

Number of Instantiations: 1

Segment Class Type: Operating System

****Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 1.000000E-03	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 8.000000E+03	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 1.000000E+04	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

*****Segment Class Name: CLEAN UP**

Target Processor Class: CPU

Number of Instantiations: 1

Segment Class Type: Operating System

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 3.000000E-04	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 8.000000E+03	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 4.500000E+03	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

— List Empty —

*****Segment Class Name: GET DATA**

Target Processor Class: DMA

Number of Instantiations: 4

Segment Class Type: Application Code

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 4.000000E-01	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 3.020000E+06	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00	Q1 = 1.200000E+07	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 3.000000E+06	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00	Q1 = 3.000000E+06	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: v_io_bus

Bus Variable Class Restriction: I/O BUS

R = 1.000000E+00	Q1 = 3.000000E+06	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

*****Segment Class Name: STORE DATA**

Target Processor Class: DMA

Number of Instantiations: 1

Segment Class Type: Application Code

******Transfer Functions List:**

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00	Q1 = 1.500000E-01	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 3.020000E+06	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00	Q1 = 1.200000E+07	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: v_local_mem

Memory Variable Class Restriction: LOCAL MEMORY

R = 1.000000E+00	Q1 = 1.000000E+06	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Memory Variable Name: v_global_mem

Memory Variable Class Restriction: GLOBAL MEMORY

R = 1.000000E+00	Q1 = 1.000000E+06	L1 = 0.000000E+00
	Q2 = 0.000000E+00	L2 = 0.000000E+00
	Q3 = 0.000000E+00	

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: v_io_bus

Bus Variable Class Restriction: I/O BUS

R = 1.000000E+00	Q1 = 1.000000E+06	L1 = 0.000000E+00

Q2 = 0.000000E+00   L2 = 0.000000E+00

Q3 = 0.000000E+00

****Thread Table**

***Thread Name: Sort/Search First Thread: CPU

Target Processor: cpu_2

Target Processor Class: CPU

******Segment List:**

Segment Name: Thread 1 - Segment 1

Segment Class: SYSTEM INITIALIZATION

Segment Type: Operating System

Segment Name: Thread 1 - Segment 2

Segment Class: CPU JOIN

Segment Type: Join

Predecessor Segment: Thread 2 - Segment 2

Predecessor Thread: Sort/Search Second Thread: DMA

Segment Name: Thread 1 - Segment 3

Segment Class: FORK PROCESS

Segment Type: Operating System

Segment Name: Thread 1 - Segment 4

Segment Class: CPU JOIN

Segment Type: Join

Predecessor Segment: Thread 2 - Segment 4

Predecessor Thread: Sort/Search Second Thread: DMA

Segment Name: Thread 1 - Segment 5
Segment Class: PROCESS DATA
Segment Type: Application Code

Segment Name: Thread 1 - Segment 6
Segment Class: CPU JOIN
Segment Type: Join
Predecessor Segment: Thread 1 - Segment 6
Predecessor Thread: Sort/Search Second Thread: DMA

Segment Name: Thread 1 - Segment 7
Segment Class: PROCESS DATA
Segment Type: Application Code

Segment Name: Thread 1 - Segment 8
Segment Class: CPU JOIN
Segment Type: Join
Predecessor Segment: Thread 2 - Segment 8
Predecessor Thread: Sort/Search Second Thread: DMA

Segment Name: Thread 1 - Segment 9
Segment Class: PROCESS DATA
Segment Type: Application Code

Segment Name: Thread 1 - Segment 10
Segment Class: CPU JOIN
Segment Type: Join
Predecessor Segment: Thead 2 - Segment 10
Predecessor Thread: Sort/Search Second Thread: DMA

Segment Name: Thread 1 - Segment 11

Segment Class: PROCESS DATA

Segment Type: Application Code

Segment Name: Thread 1 - Segment 12

Segment Class: SORT DATA

Segment Type: Application Code

Segment Name: Thread 1 - Segment 13

Segment Class: CLEAN UP

Segment Type: Operating System

*****Thread Name: Sort/Search Second Thread: DMA**

Target Processor: dma_2

Target Processor Class: DMA

******Segment List:**

Segment Name: Thread 2 - Segment 1

Segment Class: DMA JOIN

Segment Type: Join

Predecessor Segment: Thread 1 - Segment 1

Predecessor Thread: Sort/Search First Thread: CPU

Segment Name: Thread 2 - Segment 2

Segment Class: LOAD INSTRUCTIONS

Segment Type: Operating System

Segment Name: Thread 2 - Segment 3

Segment Class: DMA JOIN

Segment Type: Join

Predecessor Segment: Thread 1 - Segment 3

Predecessor Thread: Sort/Search First Thread: CPU

Segment Name: Thread 2 - Segment 4

Segment Class: GET DATA

Segment Type: Application Code

Segment Name: Thread 2 - Segment 5

Segment Class: DMA JOIN

Segment Type: Join

Predecessor Segment: Thread 1 - Segment 5

Predecessor Thread: Sort/Search First Thread: CPU

Segment Name: Thread 1 - Segment 6

Segment Class: GET DATA

Segment Type: Application Code

Segment Name: Thread 2 - Segment 7

Segment Class: DMA JOIN

Segment Type: Join

Predecessor Segment: Thread 1 - Segment 7

Predecessor Thread: Sort/Search First Thread: CPU

Segment Name: Thread 2 - Segment 8

Segment Class: GET DATA

Segment Type: Application Code

Segment Name: Thread 2 - Segment 9

Segment Class: DMA JOIN

Segment Type: Join

Predecessor Segment: Thread 1 - Segment 9

Predecessor Thread: Sort/Search First Thread: CPU

Segment Name: Thead 2 - Segment 10

Segment Class: GET DATA

Segment Type: Application Code

Segment Name: Thread 2 - Segment 11

Segment Class: DMA JOIN

Segment Type: Join

Predecessor Segment: Thread 1 - Segment 12

Predecessor Thread: Sort/Search First Thread: CPU

Segment Name: Thread 2 - Segment 12

Segment Class: STORE DATA

Segment Type: Application Code

A.3 System Load Data Structures

This is the listing for the fourTasks instantiated to obtain the System Load using System Load operations in IPERM. TheTask Class of eachTask is shown, as well as prior and subsequentTask dependencies, data set size, and initial time offset.

Load Name: TESTCASE

Author: SPARTA

Load Creation Date: 07/26/89 Validated: TRUE

Load Validation Date: 07/26/89 Load Validation Time: 14:43:14.69

Load DefinitionProcessor Ensemble Data:

Processor Ensemble Name: TEST CASE

PE Validation Date: 07/26/89 PE Validation Time: 14:26:47.52

***Task List:**

Task Name:Task Three

Task Class Name: BIG SORT/SEARCH

Processor Ensemble Name: TEST CASE

Task Minimum Start Time: 1.000000E-03

Task Data Set Size: 500

Task Class Author: SPARTA

Task Class Creation Date: 07/25/89

Task Class Validation Date: 07/26/89

Task Class Validation Time: 14:39:33.78

****Input Start Time Dependency Variables Assignments List:**

Variable Name: **==> Assigned Value:**

— List Empty —

****Output Start Time Dependency Variables Assignments List:**

Variable Name: **==> Assigned Value:**

— List Empty —

Task Name: First Ephemeris Generation

Task Class Name: EPHEMERIS GENERATION

Processor Ensemble Name: TEST CASE

Task Minimum Start Time: 0.000000E+00

Task Data Set Size: 1300

Task Class Author: SPARTA

Task Class Creation Date: 07/25/89

Task Class Validation Date: 07/26/89

Task Class Validation Time: 14:39:29.50

****Input Start Time Dependency Variables Assignments List:**

Variable Name:	**==> Assigned Value:**
v_prior_tracking	First Tracking

****Output Start Time Dependency Variables Assignments List:**

Variable Name:	**==> Assigned Value:**
v_subsequent_tracking	Second Tracking

Task Name: Second Tracking

Task Class Name: TRACKING

Processor Ensemble Name: TEST CASE

Task Minimum Start Time: 0.000000E+00

Task Data Set Size: 2600

Task Class Author: SPARTA

Task Class Creation Date: 07/24/89

Task Class Validation Date: 07/26/89

Task Class Validation Time: 14:39:23.79

****Input Start Time Dependency Variables Assignments List:**

Variable Name:	**==> Assigned Value:**
v_prior_ephemeris_gen	First Ephemeris Generation

****Output Start Time Dependency Variables Assignments List:**

Variable Name:	**==> Assigned Value:**
v_subsequent_ephem_gen	dummy

Task Name: First Tracking

Task Class Name: TRACKING

Processor Ensemble Name: TEST CASE

Task Minimum Start Time: 0.000000E+00

Task Data Set Size: 2000

Task Class Author: SPARTA

Task Class Creation Date: 07/24/89

Task Class Validation Date: 07/26/89

Task Class Validation Time: 14:39:23.79

****Input Start Time Dependency Variables Assignments List:**

Variable Name:	**==> Assigned Value:**
v_prior_ephemeris_gen	dummy

****Output Start Time Dependency Variables Assignments List:**

Variable Name:	==> Assigned Value:
v_subsequent_ephem_gen	First Ephemeris Generation

APPENDIX B

TRANSFER FUNCTION DESCIPTIONS

PARAMETERS

Many parameters were used in deriving the transfer equations. Each of these parameters, along with its assigned value for this scenario, is described on the following pages. The parameters are divided into five categories: input, data sizes, other parameters, scenario dependent, and timing estimates. Also included at the end of this section is a list of derived expressions. These are provided to simplify the interpretation of the transfer equation descriptions.

Input: Data size is the only random input variable allowed in PERM.

N	=	the number of objects to track

Data Sizes: These message and data file sizes were obtained from the Boeing Airborne Optical Adjunct Mission Data Processor Algorithm Design Document, 30 January 1987. The sizes are expressed in words.

BINPTR	=	Bin Pointer = 1
BUFFER1	=	Size of buffer on M1 = 286
BUFFER2	=	Size of buffer on M2 = 856602
BUFFER3	=	Size of buffer on M3 = 52671
BUFFER4	=	Size of buffer on M4 = 251064
BUFFER5	=	Size of buffer on M5 = 251064
CTM	=	Candidate Track Message = 92
ELCOR	=	Elevation Correction Message = 10
FTLF	=	Failed Track List File record = 5
HTSM	=	Handover Track Status Message = 19
INA	=	Initiator Message = 1
INR	=	Initiator Request Message = 5
MPNMEM	=	Measurement Processing Nodal Memory record = 23
MSM	=	Missed Sighting Message = 3
NHF	=	Navigational History File record = 105
OIF	=	Object Irradiance File record = 16
OSDF	=	Object State Data File record = 16

OSR	=	Object Sighting Report = 23
OTF	=	Object Track File record = 236
PCTM	=	Partial Candidate Track Message = 46
PRH	=	Prediction to Handover Message = 104
PWF	=	Predicted Window File record = 18
SSM	=	Star Sighting Message = 18
SSOS	=	Sensor Start of Scan Message = 13
TAM	=	Track Accept Message = 9
TARM	=	Track Accept/Reject Message = 9
TDRM	=	Track Dropped Message = 1
TUM	=	Track Update Message = 19

Instruction Set Sizes: These are the instruction set sizes in words of each major function of the Airborne Optical Adjunct Mission Data Processor. These sizes were estimated from the current Boeing AOA MDP software version as of April 1989.

ARSIS	=	Angular Rate Smoothing instruction set size = 13056
CGPIS	=	Candidate Generation Process instruction set size = 21248
DESIS	=	Designation instruction set size = 3840
HOIS	=	Handover instruction set size = 15104
MPIS	=	Measurement Processing instruction set size = 9216
NAVIS	=	Navigational Update instruction set size = 8704
OSCIS	=	Object Screening instruction set size = 15872
OSOIS	=	Object Sorting instruction set size = 6144
PRIS	=	Prediction instruction set size = 45312
PWFIS	=	Predicted Window File Sort instruction set size = 3584
RDIIS	=	Radiometric Initialization instruction set size = 21760
RDUIS	=	Radiometric Update instruction set size = 17664
RSMIS	=	Reference Star Matching instruction set size = 14848
TDMIS	=	Track Data Management instruction set size = 15872
TFIS	=	Trajectory Fitting instruction set size = 22272
TIIS	=	Track Initialization instruction set size = 28928
TUIS	=	Track Update instruction set size = 21760

Other Parameters: These are other parameters defined in the Boeing Airborne Optical Adjunct Mission Data Processor Algorithm Design Document.

AOAPC	=	average Airborne Optical Adjunct architecture polling cycle difference from the Advanced Onboard Signal Processor architecture = 0.0000055 seconds
BINS	=	number of bins = (classified, For this scenario, an unclassified estimation of 88 will be used.)
CII	=	number of initial initiators processed by CGP Lag = 3
FRAMES	=	number of frames retained = 8
NAVUP	=	number of times NAV updated per frame = 10
PSL	=	percent of stars processed in CGP Lag = 0.05
TII	=	number of initial initiators processed by TDM Initial Initiator Loading = 3
TII2	=	number of initiators processed by TI Lag = 3

Scenario Dependent: These are parameters that are dependent either on the threat or scenario chosen by the analyst.

ENDOC	=	percent of correlated objects that are Endo = 0.1
EXOC	=	percent of correlated objects that are Exo = 0.9
ENDOT	=	percent of total accepted tracks that are Endo = 0.1
EXOT	=	percent of total accepted tracks that are Exo = 0.9
PBIN1	=	percent of N objects in first bin = 0.05
PDT	=	percent of N objects that are dropped tracks = 0.03
PFCI	=	probability of failure in CGPI = 0.3
PFCII	=	probability of failure in CGPII = 0.2
PFCIII	=	probability of failure in CGPIII = 0
PFTF	=	probability of failure in Track Fitting = 0.2
PFTI	=	probability of failure in Track Initialization = 0.3
PFS	=	percent of N objects falsely classified as stars = 0.02
PLC	=	percent of correlated objects that are lethal = 0.2
PLT	=	percent of total accepted tracks that are lethal = 0.2
POC	=	percent of objects correlated = 0.7
POMS	=	percent of N that are missed sightings = 0.03

PTACT	=	previous window's total accepted tracks (assumed same as TACT) = 0.203347N
STARS	=	number of stars identified = 30
TN	=	number of track nodes = 2

Timing Estimates: These were extracted from a United States Army Strategic Defense Command timing study of the Airborne Optical Adjunct (AOA) Mission Data Processor (MDP), AOA-DP-SIM MDP Simulation Final Report, Teledyne Brown Engineering, 12 April 1988. This is a list of the times in seconds required to process each of the specific paths of the AOA MDP software. The scheduling execution/overhead costs are included in these estimates.

ARSCOST	=	Angular Rate Smoothing cost = 0.0004
CGP1COST	=	cost for CGP1 = 0.082713
CGP2COST	=	cost for CGP2 = 0.082713
CGP3COST	=	cost for CGP3 = 0.082713
CGPCOST5	=	CGP Track Accept Message Handling cost = 0.0001
DESICOST	=	Designation Initialization cost = 0.005
DESUCOST	=	Designation Update cost = 0.0035
HOCOST1	=	Handover Track Status Message processing cost = 0.0005
HOCOST2	=	Low Handover processing cost = 0.0005
MPCOST1	=	Measurement Processing cost for objects = 0.0004
MPCOST2	=	Measurement Processing cost for stars = 0.0008
NAVCOST	=	Navigation Update cost = 0.05
OSOCOST2	=	Object Sorting end of scan processing cost = 0.00002
OSOCOST3	=	Object Sorting data movement cost = 0.0001
PRICOST1	=	Prediction Initialization cost for Exo objects = 0.02
PRICOST2	=	Prediction Initialization cost for Endo objects = 0.02
PRICOST3	=	Prediction Initialization cost for Impact Point Prediction = 0.12
PRUCOST1	=	Prediction Update cost for Exo objects = 0.017
PRUCOST2	=	Prediction Update cost for Endo objects = 0.024
PRUCOST3	=	Prediction Update cost for Impact Point Prediction = 0.12
PWFCOST	=	Predicted Window File Sort cost = 0.00005
RDICOST	=	Radiometric Discriminant Initialization cost = 0.02
RDUCOST1	=	Radiometric Update cost for Endo objects = 0.007

RDUCOST2 = Radiometric Update cost for Exo objects = 0.007
RSMCOST = Reference Star Matching cost = 0.2
TDMCOST1 = TDM cost for initial initiator loading = 0.0001
TDMCOST2 = TDM cost for remaining initiator loading = 0.0001
TDMCOST3 = TDM cost for track accept handling = 0.0002
TDMCOST4 = TDM cost for track reject handling = 0.0002
TDMCOST5 = TDM cost for building candidate track messages = 0.0001
TFCOST = Track Fitting cost = 0.035
TICOST = Track Initialization cost = 0.096
TUCOST1 = Track Update cost for Endo objects = 0.0058
TUCOST2 = Track Update cost for Exo objects = 0.0058

The following timing estimates were derived by estimating the number of instructions from algorithms in the AOA MDP Algorithm Design document. These instructions were converted to times in seconds by multiplying each type of instruction by the AOSP clock speed and the number of clock cycles required to execute that instruction. The number of clock cycles were based on the Honeywell Generic VHSIC Spaceborne Computer, RH-1750.

OSCCOST = Object Screening Cost = $0.0000079N^2 + 0.000001942N + 0.00000033$

OSOCOST1 = Object Sorting cost for sorting objects into bins = $0.00000018N^2 + 0.000004N$

Derived Expressions: These are provided in this documentation to simplify the interpretation of the transfer functions.

C2 = number of initiators that pass through CGPII
$= N \times (1 - POC) \times [(1 - PFCI) \times PFTF + PFCI]$
$= N \times 0.3 \times [0.7 \times 0.2 + 0.3]$
$= 0.132N$

C3 = number of initiators that pass through CGPIII
$= A \times [PFCII + (1 - PFCII) \times PFTF]$
$= 0.132N \times [0.2 + 0.8 \times 0.2]$

= 0.04752N

CT = number of candidate tracks
= N x (1 - POC) x (1 - PFCI) + A x (1 - PFCII) + C x (1 - PFCIII)
= N x 0.3 x 0.7 + 0.132N x 0.8 + 0.04752N x 1
= 0.36312N

FT = number of failed tracks
= CT x [PFTF + (1 - PFTF) x PFTI]
= 0.36312N x [0.2 + 0.8 x 0.3]
= 0.1597728N

MEM* = previsous amount of data on the specific nodal memory * (*=1,2,3,4, or 5). This parameter's value may change from segment to segment.

IR = number of remaining initiators
= N x (1 - POC) - TN x CII + A + C
= N x 0.3 - 2 x 3 + 0.132N + 0.04752N
= 0.3N - 6 - 0.132N + 0.04752N
= 0.21552N - 6

TACT = total accepted tracks
= CT - FT
= 0.36312N - 0.1597728N
= 0.203347N

TFAC = number of tracks accepted in Track Fitting
= (1 - PFTF) x CT
= 0.8 x 0.36312N
= 0.290496N

SEGMENT DESCRIPTIONS AND TRANSFER EQUATIONS

The segment descriptions on the following pages are listed in order of the segment numbers. The Runtime equations are given in seconds; and the Memory Usage, Memory Access, and Bus Access equations are given in words. There are two bytes per word.

Segment: **Object Sorting Lag (1)**

Processor: P1

Function: Sort objects into azimuth and elevation bins and output to Object Screening. Sorting Lag represents the sorting of the first bin.

Runtime:

RUNTIME = PBIN1 x (OSOCOST1 + OSOCOST2 + OSOCOST3 x N)

= $0.05 \times [(0.00000018N^2 + 0.000004N) + 0.00002 + 0.0001N]$

= $\mathbf{0.0000009N^2 + 0.0000052N + 0.000001}$ **(seconds)**

Memory Access: Decrementing MPNMEM

M1_A = PBIN1 x N x MPNMEM

= 0.05 x N x 23 + 6144

= **1.15N (words)**

Memory Usage:

(none)

Bus Access:

(OSR to OSC)

B3 = PBIN1 x N x OSR + BINPTR

= 0.05 x N x 23 + 1

= **1.15N + 1 (words)**

Segment: **Object Sorting (2)**

Processor: P1

Function: Sort objects into azimuth and elevation bins and output to Object Screening. Object Sorting represents the processing of all bins after the first.

Runtime:

RUNTIME = (1 - PBIN1) x (OSOCOST1 + OSOCOST2 + OSOCOST3 x N)
= 0.95 x [($0.00000018N^2$ + 0.000004N) + 0.00002 + 0.0001N]
= **$0.00000018N^2$ + 0.000099N + 0.000019**

Memory Access: Decrementing MPNMEM & Loading Instruction Set

M1_A = (1 - PBIN1) x N x MPNMEM + OSOIS
= 0.95 x N x 23 + 6144
= **21.85N + 6144**

Memory Usage:

(none)

Bus Access:

(OSR to OSC)

B 3 = (1 - PBIN1) x N x OSR + BINS x BINPTR
= 0.95 x N x 23 + BINS x 1
= 21.85N + BINS
= **21.85N + 88**

Segment: **Object Screening Lag (3)**

Processor: P2

Function: For the first bin of all objects, correlate object sightings with established tracks and send correlated objects to Track Update.

Runtime:

RUNTIME = PBIN1 x OSCOST

= $0.05 \times (0.0000079N^2 + 0.000001942N + 0.00000033)$

= $\mathbf{0.0000004N^2 + 0.0000009N + 0.00000017}$

Memory Access: Decrementing PWF

M2_A = PTACT x PBIN1 x PWF

= 0.203347N x 0.05 x 18

= **0.1830123N**

Memory Usage:

(Decrement PWF)

M2 = - (PWF x PTACT x PBIN1) + MEM2

= - (18 x 0.203347N x 0.05) + 0.203347N + 134090

= **0.0203347N + 134090**

Bus Access:

(MSM to PR; TUM, TDRM to TU)

B1, B2 = (1/TN) x [PBIN1 x N x (TUM x POC + POMS x MSM + PDT x TDRM)]

= 0.5 x [0.05 x N x (19 x 0.7 + 0.03 x 3 + 0.03 x 1)]

= **0.3355N**

Segment: **Object Screening (4)**

Processor: P2

Function: For all bins after the first, correlate object sightings with established tracks and send correlated objects to Track Update.

Runtime:

RUNTIME = (1 - PBIN1) x OSCOST

= 0.95 x ($0.0000079N^2$ + 0.000001942N + 0.00000033)

= **$0.0000075N^2$ + 0.0000019N + 0.00000033**

Memory Access: Decrementing PWF & Loading Instruction Set

M2_A = PTACT x (1 - PBIN1) x PWF + OSCIS

= 0.203347N x 0.95 x 18 + 15872

= **3.4772346N + 15872**

Memory Usage:

(Decrement PWF)

M2 = - (PWF x PTACT x (1 - PBIN1)) + MEM2

= - (18 x 0.203347N x 0.95) + 0.0203347N + 134090

= **-3.456899N + 134090**

Bus Access:

(MSM to PR; TUM, TDRM to TU)

B1, B2 = (1/TN) x [(1 - PBIN1) x N x (TUM x POC + POMS x MSM + PDT x TDRM)]

= 0.5 x [0.95 x N x (19 x 0.7 + 0.03 x 3 + 0.03 x 1)]

= **6.3745N**

Segment: **Angular Rate Smoothing (5)**

Processor: P3

Function: Adjust uncorrelated object rates by averaging all of the uncorrelated object rates, for all bins except the first.

Runtime:

RUNTIME = (1 - PBIN1) x N x ARSCOST x (1 - POC) - ACCESSES x AOAPC
= 0.95 x N x 0.0004 x 0.3 - 0.855N x 0.0000055
= **0.0001093N**

where ACCESSES = number of memory accesses
= (1-PBIN1) x N x (1-POC) x 3 = 0.85N

Memory Access: Incrementing OSDF & OIF & Loading Instruction Set

(Incrementing OIF)

M2_A = (1 - PBIN1) x N x (1 - POC) x OIF x FRAMES + ARSIS
= 0.95 x N x 0.3 x 16 x 8 + 13056
= **36.48N + 13056**

(Incrementing OSDF)

M4_A,M5_A = (1-PBIN1) x N x (1-POC) x OSDF x FRAMES
= 0.95 x N x 0.3 x 16 x 8
= **36.48N**

Memory Usage:

(Increment OSDF)

M4, M5 = N x OSDF x (1 - POC) x FRAMES
= N x 16 x 0.3 x 8
= **38.4N**

(Increment OIF)

M2 = N x OIF x (1 - POC) x FRAMES
= N x 16 x 0.3 x 8
= **38.4N**

Bus Access:

(OSR to CGP)

$$B1,\ B2 = (1 - PBIN1) \times N \times OSR \times (1 - POC)$$
$$= 0.95 \times N \times 23 \times 0.3$$
$$= 6.555N$$

Segment: **Track Data Manager Initial Initiator Loading (6)**

Processor: P3

Function: Load each track (Candidate Generation Process) node with three track initiators.

Runtime:

RUNTIME = TDMCOST1 x TN x TII - ACCESSES x AOAPC
= 0.0001 x 2 x 3 - 0 x 0.0000055
= **0.0006**

where ACCESSES = number of memory accesses
= 0

Memory Access: Loading Instruction Set
(none)

Memory Usage:

M2 = MEM2
= **1.92N**

Bus Access:
(INA to CGP)

B1, B2 = TII x INA
= 3 x 1
= **3**

Segment: **Track Data Manager Build Candidate Track Messages (7)**

Processor: P3

Function: Receive partial candidate track messages from the Candidate Track Generation Process and build Candidate Track Messages. Route candidate tracks to a track node.

Runtime:

RUNTIME = TDMCOST5 x CT - ACCESSES x AOAPC
= 0.0001 x 0.36312N - 0.36312N x 0.0000055
= **0.0000343N**
where ACCESSES = number of memory accesses
= CT = 0.36312N

Memory Access: Reading OIF

M2_A = CT x OIF x FRAMES
= 0.36312N x 16 x 8
= **46.47936N**

Memory Usage:

M2 = MEM2
= **38.603347N + 134090**

Bus Access:

(CTM to TF)

B1, B2 = (1/TN) x CTM x CT
= 0.5 x 92 x 0.36312N
= **16.70352N**

Segment: **Track Data Manager Remaining Initiator Loading (8)**

Processor: P3

Function: Load the track (Candidate Generation Process) node with remaining track initiators.

Runtime:

RUNTIME = TDMCOST2 x IR - ACCESSES x AOAPC
= 0.0001 x (0.21552N - 6) - 0 x 0.0000055
= **0.0000216N - 0.0006**
where ACCESSES = number of memory accesses
= 0

Memory Access:

(none)

Memory Usage:

M2 = MEM2
= **38.603347N + 134090**

Bus Access:

(INA to CGP)

B1, B2 = (1/TN) x INA x IR
= 0.5 x 1 x (0.21552N - 6)
= **0.10776N - 3**

Segment: **Track Data Manager Track Accept/Reject Message Handler (9)**

Processor: P3

Function: Process the track accept or reject messages from track initialization. If accepted, then a track accept message must be routed to the Candidate Generation Process. If rejected, send initiator back to the Candidate Generation Process.

Runtime:

RUNTIME = TDMCOST3 x TFAC + TDMCOST4 x FT - ACCESSES x AOAPC
= 0.0002 x 0.290496N + 0.0002 x 0.1597728N - 0.3631198N x 0.0000055
= **0.0000881N**
where ACCESSES = number of memory accesses
= FT + TACT = 0.3631198N

Memory Access: Incrementing FTLF & Decrementing OIF & Loading Instruction Set

M2_A = FT x FTLF + TACT x OIF x FRAMES + TDMIS
= 0.1597728N x 5 + 0.203347N x 16 x 8 + 15872
= **26.82728N + 15872**

Memory Usage:

(Increment FTLF & Decrement OIF)

M2 = FTLF x FT - OIF x FRAMES x TACT + MEM2
= 5 x 0.1597728N - 16 x 8 x 0.203347N + 38.603347N + 134090
= **13.373795N + 134090**

Bus Access:

(TAM to CGP)

B1, B2 = TFAC x TAM
= 0.290496N x 9
= **2.614464N**

Segment: **Predicted Window File Sort (10)**

Processor: P3

Function: The track's Predicted Window File must be sorted into azimuth and elevation bins prior to use in Object Screening.

Runtime:

RUNTIME = PWFCOST x TACT - ACCESSES x AOAPC

= 0.00005 x 0.203347N - 0.203347N x 0.0000055

= **0.000009N**

where ACCESSES = number of memory accesses

= TACT = 0.203347N

Memory Access: Incrementing PWF & Loading Instruction Set

M2_A = TACT x PWF + PWFIS

= 0.203347N x 18 + 3584

= **3.660246N + 3584**

Memory Usage:

(Increment PWF)

M2 = PWF x TACT + MEM2

= 18 x 0.203347N + 13.373795N + 134090

= **17.034041N + 134090**

Bus Access:

(none)

Segment: **Candidate Generation Process Lag (11)**

Processor: P4

Function: Generate candidate tracks for all initiators initially loaded on all track nodes.

Runtime:

RUNTIME = CII x CGP1COST - ACCESSES x AOAPC
= 3 x 0.082713 - 3 x 0.0000055
= **0.2481225**

where ACCESSES = number of memory accesses
= CII = 3

Memory Access: Reading OSDF

M4_A = CII x OSDF x FRAMES
= 3 x 16 x 8
= **384**

Memory Usage:

(none)

Bus Access:

(INR, PCTM to TDM)

B1 = CII x INR + CII x (1 - PFCI) x PCTM
= 3 x 5 + 3 x 0.7 x 46
= **111.6**

Segment: **Candidate Generation Process (12)**

Processor: P4

Function: Generate candidate tracks for all initiators remaining after the initially loaded initiators are processed.

Runtime:

RUNTIME = CGP1COST x [(1/TN) x N (1 - POC) - CII] + CGP2COST x C2 + CGP3COST x C3 - ACCESSES x AOAPC

= 0.082713 x [0.5 x N x 0.3 - 3] + 0.082713 x 0.132N + 0.082713 x 0.04752N - (0.47952N - 3) x 0.0000055

= **0.027253N - 0.2481225**

where ACCESSES = number of memory accesses

= N x (1-POC) - CII + C2 + C3 = 0.47952N - 3

Memory Access: Reading OSDF & Loading Instruction Set

M4_A = (N x (1 - POC) - CII + C2 + C3) x OSDF x FRAMES + CGPIS

= (N x 0.3 - 3 + 0.132N + 0.04752N) x 16 x 8 + 21248

= **61.37856N + 20864**

Memory Usage:

(none)

Bus Access:

(INR, PCTM to TDM)

B1 = (1/TN) x [INR x IR + PCTM x [[N x (1 - POC) - CII x TN] x (1 - PFCI) + C2 x (1 - PFCII) + C3 x (1 - PFCIII)]]

= 0.5 x [5 x (0.21552N - 6) + 46 x [[N x 0.3 - 3 x 2] x 0.7 + 0.132N x 0.8 + 0.04752N x 1]]

= **8.89056N - 111.6**

Segment: **Candidate Generation Process Track Accept Message Handler (13)**

Processor: P4

Function: Remove all of the object sighting records in the track from the uncorrelated database so that they are not used in other tracks.

Runtime:

RUNTIME = CGPCOST5 x TACT - ACCESSES x AOAPC
= 0.0001 x 0.203347N - 0.203347N x 0.0000055
= **0.0000192N**
where ACCESSES = number of memory accesses
= TACT = 0.203347N

Memory Access: Decrementing OSDF

M4_A = TACT x OSDF x FRAMES
= 0.203347N x 16 x 8
= **26.028416N**

Memory Usage:

(none)

Bus Access:

(none)

Segment: **Track Fitting (14)**

Processor: P5

Function: Determine which candidate tracks are valid ballistic trajectories.

Runtime:

RUNTIME = 1/TN x CT x TFCOST - ACCESSES x AOAPC
= 0.5 x 0.36312N x 0.035 - 0 x 0.0000055
= **0.0063546N**

where ACCESSES = number of memory accesses
= 0

Memory Access: Loading Instruction Set

M4_A = TFIS
= **22272**

Memory Usage:

(none)

Bus Access:

(TARM to TDM)

B 1 = 1/TN x CT x TARM
= 0.5 x 0.36312N x 9
= **1.63404N**

Segment: **Track Initialization (All) (15)**

Processor: P5

Function: Establish track files for those tracks passing the validation test.

Runtime:

RUNTIME = TICOST x TFAC x 1/TN - ACCESSES x AOAPC
= 0.096 x 0.290496N x 0.5 - 0 x 0.0000055
= **0.0139438N**
where ACCESSES = number of memory accesses
= 0

Memory Access: Loading Instruction Set

M4_A = TIIS
= **28928**

Memory Usage:

(none)

Bus Access:

(HTSM to HO)

B 5 = HTSM x TFAC x 1/TN
= 19 x 0.290496N x 0.5
= **2.759712N**

(TARM to TDM)

B 1 = TARM x TFAC x 1/TN
= 9 x 0.290496N x 0.5
= **1.307232N**

Segment: **Radiometric Initialization (16)**

Processor: P5

Function: Compute the initial values for the radiometric features of the new track.

Runtime:

RUNTIME = RDICOST x TACT x 1/TN - ACCESSES x AOAPC
= 0.02 x 0.2033472N x 0.5 - 0.1016735N x 0.0000055
= **0.0020329N**
where ACCESSES = number of memory accesses
= 1/TN x TACT = 0.1016735N

Memory Access: Incrementing OTF & Loading Instruction Set

M4_A = 1/TN x TACT x OTF + RDIIS
= 0.5 x 0.2033472N x 236 + 21760
= **23.994946N + 21760**

Memory Usage:

(Increment OTF)

M4 = OTF x TACT x 1/TN + MEM4
= 236 x 0.203347N x 0.5 + 38.3895N + 483000
= **62.384446N + 483000**

Bus Access:

(none)

Segment: **Designation Initialization (17)**

Processor: P5

Function: Use radiometric and metric data to designate objects. All objects are designated as either lethal or non-lethal.

Runtime:

RUNTIME = DESICOST x TACT x 1/TN - ACCESSES x AOAPC
= 0.005 x 0.203347N x 0.5 - 0.203347N x 0.0000055
= **0.0005072N**
where ACCESSES = number of memory accesses
= 2 x TACT x 1/TN = 0.203347N

Memory Access: Reading & Writing OTF & Loading Instruction Set

M4_A = 2 x TACT x 1/TN x OTF + DESIS
= 2 x 0.203347N x 0.5 x 236 + 3840
= **47.98824N + 3840**

Memory Usage:

M4 = MEM4
= **62.384446N + 483000**

Bus Access:

(none)

Segment: **Prediction Initialization (18)**

Processor: P5

Function: Compute target position handover estimates, target prediction handover estimates, and impact point prediction handover estimates.

Runtime:

RUNTIME = (1/TN) x [PRICOST1 x TACT x EXOT + PRICOST2 x ENDOT x TACT + PRICOST3 x PLT x TACT x ENDOT] - ACCESSES x AOAPC

= 0.5 x [0.02 x 0.203347N x 0.9 + 0.02 x 0.1 x 0.203347N + 0.12 x 0.2 x 0.203347N x 0.1] - 0.0101674N x 0.0000055

= **0.0022775N**

where ACCESSES = number of memory accesses

= ENDOT x TACT x 1/TN = 0.0101674N

Memory Access: Reading OTF & Loading Instruction Set

M4_A = ENDOT x TACT x 1/TN x OTF + PRIS

= 0.1 x 0.203347N x 0.5 x 236 + 45312

= **2.3995064N + 45312**

Memory Usage:

M4 = MEM4

= **62.384446N + 483000**

Bus Access:

(HTSM, PRH to HO)

B5 = HTSM x 1/TN x TACT + PRH x PLT x ENDOT x TACT x 1/TN

= 19 x 0.5 x 0.203347N + 104 x 0.2 x 0.1 x 0.203347N x 0.5

= **2.1432795N**

(PWF to OSC)

B1 = PWF x TACT x 1/TN

= 104 x 0.203347N x 0.5

= **10.574054N**

Segment: **Track Update (19)**

Processor: P8

Function: Update the Object Track File for all existing tracks for which a new sighting is correlated.

Runtime:

RUNTIME = (1/TN) x [TUCOST1 x N x POC x ENDOC + TUCOST2 x N x POC x EXOC] - ACCESSES x AOAPC

= 0.5 x [0.0058 x N x 0.7 x 0.1 + 0.0058 x N x 0.7 x 0.9] - 0.7105 x 0.0000055

= **0.0020261N**

where ACCESSES = number of memory accesses

= PDT x POC x N x 1/TN + 2 x N x POC x 1/TN

= 0.7105N

Memory Access: Decrementing, Reading, & Writing OTF & Loading Instruction Set

M4_A = (PDT x POC x N x 1/TN + 2 x N x POC x 1/TN) x OTF + TUIS

= (0.03 x 0.7 x N x 0.5 + 2 x N x 0.7 x 0.5) x 236 + 21760

= **167.678N + 21760**

Memory Usage:

(Decrement OTF)

M4 = - (PDT x POC x N x 1/TN) + MEM4

= - (0.03 x 0.7 x N x 0.5) + 483000

= **-0.0105N + 483000**

Bus Access:

(HTSM to HO)

B5 = HTSM x N x POC x 1/TN

= 19 x N x 0.7 x 0.5

= **6.65N**

Segment: **Radiometric Update (20)**

Processor: P8

Function: Update the radiometric features of the new tracks.

Runtime:

RUNTIME = (1/TN) x [RDUCOST1 x N x POC x ENDOC + RDUCOST2 x N x POC x EXOC] - ACCESSES x AOAPC

= 0.5 x [0.007 x N x 0.7 x 0.1 + 0.007 x N x 0.7 x 0.9] - 0.7N x 0.0000055

= **0.0024462N**

where ACCESSES = number of memory accesses

= 2 x N x POC x 1/TN = 0.7N

Memory Access: Reading & Writing OTF & Loading Instruction Set

M4_A = 2 x N x POC x 1/TN x OTF + RDUIS

= 2 x N x 0.7 x 0.5 x 236 + 17664

= ***165.2N + 17664***

Memory Usage:

M4 = MEM4

= **38.3895N + 483000**

Bus Access:

(none)

Segment: **Designation Update (21)**

Processor: P8

Function: Use radiometric and metric data to designate objects. All objects are designated as either lethal or non-lethal.

Runtime:

RUNTIME = 1/TN x DESUCOST x N x POC - ACCESSES x AOAPC

= 0.5 x 0.0035 x N x 0.7 - 0.7N x 0.0000055

= **0.0012212N**

where ACCESSES = number of memory accesses

= 1/TN x TN x POC x 2 = 0.7N

Memory Access: Reading & Writing OTF & Loading Instruction Set

M4_A = 1/TN x N x POC x 2 x OTF + DESIS

= 0.5 x N x 0.7 x 2 x 236 + 3840

= **165.2N + 3840**

Memory Usage:

M4 = MEM4

= **38.3895N + 483000**

Bus Access:

(none)

Segment: **Prediction Update (22)**

Processor: P8

Function: Compute target position handover estimates, target prediction handover estimates, and impact point prediction handover estimates.

Runtime:

RUNTIME = (1/TN) x [PRUCOST1 x EXOC x N x POC + [PRUCOST2 x ENDOC x N x POC + PRUCOST3 x PLC x N x POC x ENDOC - ACCESSES x AOAPC

= 0.5 x (0.017 x 0.9 x N x 0.7 + 0.24 x 0.1 x N x 0.7 + 0.12 x 0.2 x N x 0.7 x 0.1) - 0.315N x 0.0000055

= **0.0145933N**

where ACCESSES = number of memory accesses

= 1/TN x ENDOC x N x POC = 0.315N

Memory Access: Reading OTF & Loading Instruction Set

M4_A = 1/TN x ENDOC x N x POC x OTF + PRIS

= 0.5 x 0.9 x N x 0.7 x 236 + 45312

= **74.34N + 45312**

Memory Usage:

M4 = MEM4

= **38.3895N + 483000**

Bus Access:

(HTSM, PRH to HO)

B 5 = HTSM x 1/TN x N x POC + PRH x PLC x ENDOC x N x POC x 1/TN

= 19 x 0.5 x N x 0.7 + 104 x 0.2 x 0.1 x N x 0.7 x 0.5

= **7.378N**

(PWF to OSC)

B 1 = PWF x N x POC x 1/TN

= 104 x N x 0.7 x 0.5

= **36.4N**

Segment: **Candidate Generation Process Lag (23)**

Processor: P6

Function: Generate candidate tracks for all initiators initially loaded on all track nodes.

Runtime:

RUNTIME = CII x CGP1COST - ACCESSES x AOAPC
= 3 x 0.082713 - 3 x 0.0000055
= **0.2481225**

where ACCESSES = number of memory accesses
= CII = 3

Memory Access: Reading OSDF

M5_A = CII x OSDF x FRAMES
= 3 x 16 x 8
= **384**

Memory Usage:

(none)

Bus Access:

(INR, PCTM to TDM)

B2 = CII x INR + CII x (1 - PFCI) x PCTM
= 3 x 5 + 3 x 0.7 x 46
= **111.6**

Segment: **Candidate Generation Process (24)**

Processor: P6

Function: Generate candidate tracks for all initiators remaining after the initially loaded initiators are processed.

Runtime:

RUNTIME = CGP1COST x [(1/TN) x N (1 - POC) - CII] + CGP2COST x C2 + CGP3COST x C3 - ACCESSES x AOAPC

= 0.082713 x [0.5 x N x 0.3 - 3] + 0.082713 x 0.132N + 0.082713 x 0.04752N - (0.47952N - 3) x 0.0000055

= **0.027253N - 0.2481225**

where ACCESSES = number of memory accesses

= N x (1-POC) - CII + C2 + C3 = 0.47952N - 3

Memory Access: Reading OSDF & Loading Instruction Set

M5_A = (N x (1 - POC) - CII + C2 + C3) x OSDF x FRAMES + CGPIS

= (N x 0.3 - 3 + 0.132N + 0.04752N) x 16 x 8 + 21248

= **61.37856N + 20864**

Memory Usage:

(none)

Bus Access:

(INR, PCTM to TDM)

B 2 = (1/TN) x [INR x IR + PCTM x [[N x (1 - POC) - CII x TN] x (1 - PFCI) + C2 x (1 - PFCII) + C3 x (1 - PFCIII)]]

= 0.5 x [5 x (0.21552N - 6) + 46 x [[N x 0.3 - 3 x 2] x 0.7 + 0.132N x 0.8 + 0.04752N x 1]]

= **8.89056N - 111.6**

Segment: **Candidate Generation Process Track Accept Message Handler (25)**

Processor: P6

Function: Remove all of the object sighting records in the track from the uncorrelated database so that they are not used in other tracks.

Runtime:

RUNTIME = CGPCOST5 x TACT - ACCESSES x AOAPC

= 0.0001 x 0.203347N - 0.203347N x 0.0000055

= **0.0000192N**

where ACCESSES = number of memory accesses

= TACT = 0.203347N

Memory Access: Decrementing OSDF

M5_A = TACT x OSDF x FRAMES

= 0.203347N x 16 x 8

= **26.028416N**

Memory Usage:

(none)

Bus Access:

(none)

Segment: **Track Fitting (26)**

Processor: P7

Function: Determine which candidate tracks are valid ballistic trajectories.

Runtime:

RUNTIME = 1/TN x CT x TFCOST - ACCESSES x AOAPC
= 0.5 x 0.36312N x 0.035 - 0 x 0.0000055
= **0.0063546N**
where ACCESSES = number of memory accesses
= 0

Memory Access: Loading Instruction Set

M5_A = TFIS
= **22272**

Memory Usage:

(none)

Bus Access:

(TARM to TDM)

B2 = 1/TN x CT x TARM
= 0.5 x 0.36312N x 9
= **1.63404N**

Segment: **Track Initialization (27)**

Processor: P7

Function: Establish track files for those tracks passing the validation test, except for the first few tracks processed by TI Lag.

Runtime:

RUNTIME = TICOST x [TFAC x 1/TN - TII2] - ACCESSES x AOAPC

= 0.096 x (0.290496N x 0.5 - 3) - 0 x 0.0000055

= **0.0139438N - 0.288**

where ACCESSES = number of memory accesses

= 0

Memory Access: Loading Instruction Set

M5_A = TIIS

= **28928**

Memory Usage:

(none)

Bus Access:

(HTSM to HO)

B 6 = HTSM x (TFAC x 1/TN - TII2)

= 19 x (0.290496N x 0.5 - 3)

= **2.759712N - 57**

(TARM to TDM)

B 2 = TARM x (TFAC x 1/TN - TII2)

= 9 x (0.290496N x 0.5 - 3)

= **1.307232N - 27**

Segment: **Radiometric Initialization (28)**

Processor: P7

Function: Compute the initial values for the radiometric features of the new track.

Runtime:

RUNTIME = RDICOST x TACT x 1/TN - ACCESSES x AOAPC
= 0.02 x 0.2033472N x 0.5 - 0.1016735N x 0.0000055
= **0.0020329N**
where ACCESSES = number of memory accesses
= 1/TN x TACT = 0.1016735N

Memory Access: Incrementing OTF & Loading Instruction Set

M5_A = 1/TN x TACT x OTF + RDIIS
= 0.5 x 0.2033472N x 236 + 21760
= **23.994946N + 21760**

Memory Usage:

(Increment OTF)

M5 = OTF x TACT x 1/TN + MEM5
= 236 x 0.203347N x 0.5 + 38.3895N + 483000
= **62.384446N + 483000**

Bus Access:

(none)

Segment: **Designation Initialization (29)**

Processor: P7

Function: Use radiometric and metric data to designate objects. All objects are designated as either lethal or non-lethal.

Runtime:

RUNTIME = DESICOST x TACT x 1/TN - ACCESSES x AOAPC
= 0.005 x 0.203347N x 0.5 - 0.203347N x 0.0000055
= **0.0005072N**
where ACCESSES = number of memory accesses
= 2 x TACT x 1/TN = 0.203347N

Memory Access: Reading & Writing OTF & Loading Instruction Set

M5_A = 2 x TACT x 1/TN x OTF + DESIS
= 2 x 0.203347N x 0.5 x 236 + 3840
= **47.98824N + 3840**

Memory Usage:

M5 = MEM5
= **62.384446N + 483000**

Bus Access:

(none)

Segment: **Prediction Initialization (30)**

Processor: P7

Function: Compute target position handover estimates, target prediction handover estimates, and impact point prediction handover estimates.

Runtime:

RUNTIME = (1/TN) x [PRICOST1 x TACT x EXOT + PRICOST2 x ENDOT x TACT + PRICOST3 x PLT x TACT x ENDOT] - ACCESSES x AOAPC

= 0.5 x [0.02 x 0.203347N x 0.9 + 0.02 x 0.1 x 0.203347N + 0.12 x 0.2 x 0.203347N x 0.1] - 0.0101674N x 0.0000055

= **0.0022775N**

where ACCESSES = number of memory accesses

= ENDOT x TACT x 1/TN = 0.0101674N

Memory Access: Reading OTF & Loading Instruction Set

M5_A = ENDOT x TACT x 1/TN x OTF + PRIS

= 0.1 x 0.203347N x 0.5 x 236 + 45312

= **2.3995064N + 45312**

Memory Usage:

M5 = MEM5

= **62.384446N + 483030**

Bus Access:

(HTSM, PRH to HO)

B6 = HTSM x 1/TN x TACT + PRH x PLT x ENDOT x TACT x 1/TN

= 19 x 0.5 x 0.203347N + 104 x 0.2 x 0.1 x 0.203347N x 0.5

= **2.1432795N**

(PWF to OSC)

B2 = PWF x TACT x 1/TN

= 104 x 0.203347N x 0.5

= **10.574054N**

Segment: **Track Update (31)**

Processor: P9

Function: Update the Object Track File for all existing tracks for which a new sighting is correlated.

Runtime:

RUNTIME = (1/TN) x [TUCOST1 x N x POC x ENDOC + TUCOST2 x N x POC x EXOC] - ACCESSES x AOAPC

= 0.5 x [0.0058 x N x 0.7 x 0.1 + 0.0058 x N x 0.7 x 0.9] - 0.7105 x 0.0000055

= **0.0020261N**

where ACCESSES = number of memory accesses

= PDT x POC x N x 1/TN + 2 x N x POC x 1/TN

= 0.7105N

Memory Access: Decrementing, Reading, & Writing OTF & Loading Instruction Set

M5_A = (PDT x POC x N x 1/TN + 2 x N x POC x 1/TN) x OTF + TUIS

= (0.03 x 0.7 x N x 0.5 + 2 x N x 0.7 x 0.5) x 236 + 21760

= **167.678N + 21760**

Memory Usage:

(Decrement OTF)

M5 = - (PDT x POC x N x 1/TN) + MEM5

= - (0.03 x O.7 x N x 0.5) + 483000

= **-0.0105N + 483000**

Bus Access:

(HTSM to HO)

B6 = HTSM x N x POC x 1/TN

= 19 x N x 0.7 x 0.5

= **6.65N**

Segment: **Radiometric Update (32)**

Processor: P9

Function: Update the radiometric features of the new tracks.

Runtime:

RUNTIME = (1/TN) x [RDUCOST1 x N x POC x ENDOC + RDUCOST2 x N x POC x EXOC] - ACCESSES x AOAPC

= 0.5 x [0.007 x N x 0.7 x 0.1 + 0.007 x N x 0.7 x 0.9] - 0.7N x 0.0000055

= **0.0024462N**

where ACCESSES = number of memory accesses

= 2 x N x POC x 1/TN = 0.7N

Memory Access: Reading & Writing OTF & Loading Instruction Set

M5_A = 2 x N x POC x 1/TN x OTF + RDUIS

= 2 x N x 0.7 x 0.5 x 236 + 17664

= **165.2N + 17664**

Memory Usage:

M5 = MEM5

= **38.3895N + 483000**

Bus Access:

(none)

Segment: **Designation Update (33)**

Processor: P9

Function: Use radiometric and metric data to designate objects. All objects are designated as either lethal or non-lethal.

Runtime:

RUNTIME = 1/TN x DESUCOST x N x POC - ACCESSES x AOAPC

= 0.5 x 0.0035 x N x 0.7 - 0.7N x 0.0000055

= **0.0012212N**

where ACCESSES = number of memory accesses

= 1/TN x N x POC x 2 = 0.7N

Memory Access: Reading & Writing OTF & Loading Instruction Set

M5_A = 1/TN x N x POC x 2 x OTF + DESIS

= 0.5 x N x 0.7 x 2 x 236 + 3840

= **165.2N + 3840**

Memory Usage:

M5 = MEM5

= **38.3895N + 483000**

Bus Access:

(none)

Segment: **Prediction Update (34)**

Processor: P9

Function: Compute target position handover estimates, target prediction handover estimates, and impact point prediction handover estimates.

Runtime:

RUNTIME = (1/TN) x [PRUCOST1 x EXOC x N x POC + [PRUCOST2 x ENDOC x N x POC + PRUCOST3 x PLC x N x POC x ENDOC - ACCESSES x AOAPC

= 0.5 x (0.017 x 0.9 x N x 0.7 + 0.24 x 0.1 x N x 0.7 + 0.12 x 0.2 x N x 0.7 x 0.1) - 0.315N x 0.0000055

= **0.0145933N**

where ACCESSES = number of memory accesses

= 1/TN x ENDOC x N x POC = 0.315N

Memory Access: Reading OTF & Loading Instruction Set

M5_A = 1/TN x ENDOC x N x POC x OTF + PRIS

= 0.5 x 0.9 x N x 0.7 x 236 + 45312

= **74.34N + 45312**

Memory Usage:

M5 = MEM5

= **38.3895N + 483000**

Bus Access:

(HTSM, PRH to HO)

B6 = HTSM x 1/TN x N x POC + PRH x PLC x ENDOC x N x POC x 1/TN

= 19 x 0.5 x N x 0.7 + 104 x 0.2 x 0.1 x N x 0.7 x 0.5

= **7.378N**

(PWF to OSC)

B2 = PWF x N x POC x 1/TN

= 104 x N x 0.7 x 0.5

= **36.4N**

Segment: **Angular Rate Smoothing Lag (35)**

Processor: P3

Function: Adjust uncorrelated object rates by averaging all of the uncorrelated object rates, for the first bin only.

Runtime:

RUNTIME = PBIN1 x N x ARSCOST x (1 - POC) - ACCESSES x AOAPC
= 0.05 x N x 0.0004 x 0.3 - 0.045N x 0.0000055
= **0.0000058N**
where ACCESSES = number of memory accesses
= PBIN1 x N x (1-POC) x 3 = 0.045N

Memory Access: Incrementing OSDF & OIF

(Incrementing OIF)

M2_A = PBIN1 x N x (1 - POC) x OIF x FRAMES
= 0.05 x N x 0.3 x 16 x 8
= **1.92N**

(Incrementing OSDF)

M4_A,M5_A = PBIN1 x N x (1-POC) x OSDF x FRAMES
= 0.05 x N x 0.3 x 16 x 8
= **1.92N**

Memory Usage:

(Increment OSDF)

M4, M5 = PBIN1 x N x OSDF x (1 - POC) x FRAMES
= 0.05 x N x 16 x 0.3 x 8
= **1.92N**

(Increment OIF)

M2 = PBIN1 x N x OIF x (1 - POC) x FRAMES
= 0.05 x N x 16 x 0.3 x 8
= **1.92N**

Bus Access:

(OSR to CGP)

$$\mathbf{B1,\ B2} = PBIN1 \times N \times OSR \times (1 - POC)$$
$$= 0.05 \times N \times 23 \times 0.3$$
$$= \mathbf{0.345N}$$

Segment: **Measurement Processing Lag (36)**

Processor: P10

Function: Input Object Sighting Messages, correct the received object angular position, angular rate and object irradiance measurements, and screen the data to identify stars.

Runtime:

RUNTIME = PBIN1 x N x MPCOST1 + [PSL x STARS + PBIN1 x N x PFS] x MPCOST2 - ACCESSES x AOAPC
= 0.05 x N x 0.0004 + [0.05 x 30 + 0.05 x N x 0.02] x 0.0008 - 1.5 x 0.0000055
= **0.0000208N + 0.0011918**
where ACCESSES = number of memory accesses
= PSL x STARS = 1.5

Memory Access: Decrementing MPNMEM

M1_A = PSL x STARS x MPNMEM
= 0.05 x 30 x 23
= **34.5**

Memory Usage:

(none)

Bus Access:

(SSOS to OSC)

B3 = SSOS
= **13**

(SSM to RSM)

B4 = SSM x (PSL x STARS + PBIN1 x N x PFS)
= 18 x (0.05 x 30 + 0.05 x N x 0.02)
= **0.018N + 27**

Segment: **Measurement Processing (37)**

Processor: P10

Function: Input Object Sighting Messages, correct the received object ang·ılar position, angular rate and object irradiance measurements, and screen the data to identify stars.

Runtime:

RUNTIME = (1 - PBIN1) x N x MPCOST1 + [(1 - PSL) x STARS + (1 - PBIN1) x N x PFS] x MPCOST2 - ACCESSES x AOAPC

= 0.95 x N x 0.0004 + [0.95 x 30 + 0.95 x N x 0.02] x 0.0008 - 28.5 x 0.0000055

= **0.0003952N + 0.0226433**

where ACCESSES = number of memory accesses

= (1-PSL) x STARS = 28.5

Memory Access: Decrementing MPNMEM & Loading Instruction Set

M1_A = (1 - PSL) x STARS x MPNMEM + MPIS

= 0.95 x 30 x 23 + 9216

= **9871.5**

Memory Usage:

(Decrement MPNMEM)

M1 = - [MPNMEM x (STARS + N)] + MEM1

= - (23 x (30 + N)) + 76432

= **-23N + 77121**

Bus Access:

(SSM to RSM)

B4 = SSM x [(1 - PSL) x STARS + (1 - PBIN1) x N x PFS]

= 18 x (0.95 x 30 + 0.95 x N x 0.02)

= **0.342N + 513**

Segment: **Initialization Node 3 (38)**

Processor: P11

Function: Initializes memory on Node 3: M3.

Runtime:

RUNTIME = 0

Memory Access:

(none)

Memory Usage:

(Increment BUFFER3 & NHF & Load all instruction sets)

M3 = NHF x FRAMES x NAVUP + BUFFER3 + NAVIS + HOIS + RSMIS

= 105 x 8 x 10 + 52671 + 8704 + 15104 + 14848

= **99727**

Bus Access:

(none)

Segment: **Navigation Update (39)**

Processor: P11

Function: Acquires and processes navigation data to update the Navigational History File.

Runtime:

RUNTIME = NAVCOST x NAVUP - ACCESSES x AOAPC
= 0.05 x 10 - 10 x 0.0000055
= **0.499945**
where ACCESSES = number of memory accesses
= NAVUP = 10

Memory Access: Writing NHF & Loading Instruction Set

M3_A = NAVUP x NHF + NAVIS
= 10 x 105 + 8704
= **9754**

Memory Usage:

(none)

Bus Access:

(NHF to MP)

B4 = NHF x NAVUP
= 105 x 10
= **1050**

(NHF to TU, PR)

B5, B6 = NHF x NAVUP x 2
= 105 x 10 x 2
= **2100**

Segment: **Handover (40)**

Processor: P12

Function: Stores and forwards prediction messages to the CSS, stores and forwards track status messages to the C & D, and forwards lethal handover messages to RSS.

Runtime:

RUNTIME = HOCOST1 x [TFAC + TACT + 2 x (N x POC)] + HOCOST2 x ENDOT x PLT x TACT - ACCESSES x AOAPC

= 0.0005 x [0.290496N + 0.203347N + 2 x N x 0.7] + 0.0005 x 0.1 x 0.1 x 0.203347N - 1.8979099 x 0.0000055

= **0.0009375N**

where ACCESSES = number of memory accesses

= ENDOT x PLT x TACT + TFAC + TACT + 2 x N x POC = 1.8979099N

Memory Access: Reading OTF & Loading Instruction Set

M3_A = (ENDOT x PLT x TACT + TFAC + TACT + 2 x N x POC) x OTF + HOIS

= (0.1 x 0.2 x 0.203347N + 0.290496N + 0.203347N + 2 x N x 0.7) x 236 + 15104

= **447.90674N + 15104**

Memory Usage:

(none)

Bus Access:

(none)

Segment: **Reference Star Matching (41)**

Processor: P13

Function: Accepts star sightings, matches them with stars in the reference star catalog, calculates pointing error, and transfers to SS.

Runtime:

RUNTIME = RSMCOST x (STARS + N x PFS) - ACCESSES x AOAPC

= 0.2 x (30 + N x 0.02) - 0 x 0.0000055

= **0.004N + 6**

where ACCESSES = number of memory accesses

= 0

Memory Access: Loading Instruction Set

M3_A = RSMIS

= **14848**

Memory Usage:

M3 = MEM3

= **99727**

Bus Access:

(ELCOR to TU, TF)

B5, B6 = ELCOR x 2

= 10 x 2

= **20**

Segment: **Track Initialization Lag (42)**

Processor: P7

Function: Establish track files for those tracks passing the validation test.

Runtime:

RUNTIME = TICOST x TII2 - ACCESSES x AOAPC
= 0.096 x 3 - 0 x 0.0000055
= **0.288**

where ACCESSES = number of memory accesses
= 0

Memory Access: Loading Instruction Set

(none)

Memory Usage:

(none)

Bus Access:

(HTSM to HO)

B 6 = HTSM x TII2
= 19 x 3
= **57**

B 2 = TARM x TII2
= 9 x 3
= **27**

Segment: **Initialization Node 1 (43)**

Processor: P10

Function: Initializes memory on node 1: M1.

Runtime:

RUNTIME = 0

Memory Access:

(none)

Memory Usage:

(Increment BUFFER1 & MPNMEM & Load all instruction sets)

M1 = NHF x FRAMES x NAVUP + BUFFER3 + OSOIS + MPIS

= 105 x 8 x 10 + 52671 + 6144 + 9216

= **76431**

Bus Access:

(none)

Segment: **Initialization Node 2 (44)**

Processor: P2

Function: Initializes memory on node 2: M2.

Runtime:

RUNTIME = 0

Memory Access:

(none)

Memory Usage:

(Increment BUFFER2 & PWF & Load all instruction sets)

M2 = PWF x TACT + BUFFER2 + OSCIS + ARSIS + TDMIS + PWFIS

= 104 x 0.203347N + 85602 + 15872 + 13056 + 15872 + 3584

= **0.203347N + 134090**

Bus Access:

(none)

Segment: **Initialization Node 4 (45)**

Processor: P8

Function: Initializes memory on node 4: M4.

Runtime:

RUNTIME = 0

Memory Access:

(none)

Memory Usage:

(Increment BUFFER4 & Load all instruction sets)

M4 = BUFFER4 + CGPIS + TFIS + TIIS + RDIIS + DESIS + PRIS + TUIS + RDUIS + DESIS + PRIS

= 251064 + 21248 + 22272 + 28928 + 21760 + 3840 + 45312 + 21760 + 17664 + 3840 + 45312

= **483000**

Bus Access:

(none)

Segment: **Initialization Node 5 (46)**

Processor: P9

Function: Initializes memory on node 5: M5.

Runtime:

RUNTIME = **0**

Memory Access:

(none)

Memory Usage:

(Increment BUFFER5 & Load all instruction sets)

M5 = BUFFER5 + CGPIS + TFIS + TIIS + RDIIS + DESIS + PRIS + TUIS + RDUIS + DESIS + PRIS

= 251064 + 21248 + 22272 + 28928 + 21760 + 3840 + 45312 + 21760 + 17664 + 3840 + 45312

= **483000**

Bus Access:

(none)

APPENDIX C

PERM TRACK MODEL DATA BASE

Load Name: AOA/AOSP Tracking
Author: TBE
Load Creation Date: 08/10/89 Validated: TRUE
Load Validation Date: 08/21/89 Load Validation Time: 09:42:49.08

Load Definition Processor Ensemble Data:

Processor Ensemble Name: AOSP Tracking

Author: TBE Creation Date: 07/05/89
Validated = TRUE
Validation Date: 08/10/89 Validation Time: 13:32:24.12

*Processor Class List:

**Processor Class Name: NAVIGATIONAL CONTROLLERS

***Accessible Processors Variables List:

Variable Name:	Class Restriction:
V_Local_NIOC_1	NAVIGATIONAL CONTROLLERS
V_Local_NIOC_2	NAVIGATIONAL CONTROLLERS
V_Sorter_1	SORTERS
V_Sorter_2	SORTERS
V_Screener_1	SCREENERS
V_Screener_2	SCREENERS
V_First_Tracker_1	TRACKERS
V_First_Tracker_2	TRACKERS
V_First_Tracker_3	TRACKERS
V_Second_Tracker_1	TRACKERS
V_Second_Tracker_2	TRACKERS
V_Second_Tracker_3	TRACKERS

***Accessible Memories Variables List:

Variable Name:	Class Restriction:
V_NIOC_Memory	NIOC MEMORY
V_Sort_Memory	SORT MEMORY
V_Screen_Memory	SCREEN MEMORY

***Accessible Busses Variables List:

Variable Name:	Class Restriction
V_Sort_I/O	SORT WITH NIOC
V_Screen_I/O	SCREEN WITH NIOC
V_First_Track_I/O	TRACK WITH NIOC
V_Second_Track_I/O	TRACK WITH NIOC

***Processor Class Instantiation List:

****Processor Name: P13
Processor Class: NAVIGATIONAL CONTROLLERS

*****Accessible Processors Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Local_NIOC_1	=> P11
V_Local_NIOC_2	=> P12
V_Sorter_1	=> P10
V_Sorter_2	=> P1
V_Screener_1	=> P2
V_Screener_2	=> P3
V_First_Tracker_1	=> P4
V_First_Tracker_2	=> P5
V_First_Tracker_3	=> P8
V_Second_Tracker_1	=> P6
V_Second_Tracker_2	=> P7
V_Second_Tracker_3	=> P9

*****Accessible Memories Variables Assignment List:

Variable Name:	=> Assigned Value:
V_NIOC_Memory	=> M3
V_Sort_Memory	=> M1
V_Screen_Memory	=> M2

*****Accessible Busses Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Sort_I/O	=> B4
V_Screen_I/O	=> B7

V_First_Track_I/O => B5
V_Second_Track_I/O => B6

****Processor Name: P12
Processor Class: NAVIGATIONAL CONTROLLERS

*****Accessible Processors Variables Assignment List:
Variable Name: => Assigned Value:

V_Local_NIOC_1 => P11
V_Local_NIOC_2 => P13
V_Sorter_1 => P10
V_Sorter_2 => P1
V_Screener_1 => P2
V_Screener_2 => P3
V_First_Tracker_1 => P4
V_First_Tracker_2 => P5
V_First_Tracker_3 => P8
V_Second_Tracker_1 => P6
V_Second_Tracker_2 => P7
V_Second_Tracker_3 => P9

*****Accessible Memories Variables Assignment List:
Variable Name: => Assigned Value:

V_NIOC_Memory => M3
V_Sort_Memory => M1
V_Screen_Memory => M2

*****Accessible Busses Variables Assignment List:
Variable Name: => Assigned Value:

V_Sort_I/O => B4
V_Screen_I/O => B7
V_First_Track_I/O => B5
V_Second_Track_I/O => B6

****Processor Name: P11
Processor Class: NAVIGATIONAL CONTROLLERS

*****Accessible Processors Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Local_NIOC_1	=> P12
V_Local_NIOC_2	=> P13
V_Sorter_1	=> P10
V_Sorter_2	=> P1
V_Screener_1	=> P2
V_Screener_2	=> P3
V_First_Tracker_1	=> P4
V_First_Tracker_2	=> P5
V_First_Tracker_3	=> P8
V_Second_Tracker_1	=> P6
V_Second_Tracker_2	=> P7
V_Second_Tracker_3	=> P9

*****Accessible Memories Variables Assignment List:

Variable Name:	=> Assigned Value:
V_NIOC_Memory	=> M3
V_Sort_Memory	=> M1
V_Screen_Memory	=> M2

*****Accessible Busses Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Sort_I/O	=> B4
V_Screen_I/O	=> B7
V_First_Track_I/O	=> B5
V_Second_Track_I/O	=> B6

**Processor Class Name: SCREENERS

***Accessible Processors Variables List:

Variable Name:	Class Restriction:
V_Local_Screener	SCREENERS
V_Sorter_1	SORTERS
V_Sorter_2	SORTERS

V_Navigation_1	NAVIGATIONAL CONTROLLERS
V_Navigation_2	NAVIGATIONAL CONTROLLERS
V_Navigation_3	NAVIGATIONAL CONTROLLERS
V_First_Tracker_1	TRACKERS
V_First_Tracker_2	TRACKERS
V_First_Tracker_3	TRACKERS
V_Second_Tracker_1	TRACKERS
V_Second_Tracker_2	TRACKERS
V_Second_Tracker_3	TRACKERS

***Accessible Memories Variables List:

Variable Name:	Class Restriction:
V_Screen_Memory	SCREEN MEMORY
V_First_Track_Memory	TRACK MEMORY
V_Second_Track_Memory	TRACK MEMORY

***Accessible Busses Variables List:

Variable Name:	Class Restriction
V_Sort_I/O	SORT TO SCREEN
V_NIOC_I/O	SCREEN WITH NIOC
V_First_Track_I/O	SCREEN WITH TRACK
V_Second_Track_I/O	SCREEN WITH TRACK

***Processor Class Instantiation List:

****Processor Name: P3
Processor Class: SCREENERS

*****Accessible Processors Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Local_Screener	=> P2
V_Sorter_1	=> P10
V_Sorter_2	=> P1
V_Navigation_1	=> P11
V_Navigation_2	=> P12
V_Navigation_3	=> P13
V_First_Tracker_1	=> P4

V_First_Tracker_2 => P5
V_First_Tracker_3 => P8
V_Second_Tracker_1 => P6
V_Second_Tracker_2 => P7
V_Second_Tracker_3 => P9

*****Accessible Memories Variables Assignment List:
Variable Name: => Assigned Value:

V_Screen_Memory => M2
V_First_Track_Memory => M4
V_Second_Track_Memory => M5

*****Accessible Busses Variables Assignment List:
Variable Name: => Assigned Value:

V_Sort_I/O => B3
V_NIOC_I/O => B7
V_First_Track_I/O => B1
V_Second_Track_I/O => B2

****Processor Name: P2
Processor Class: SCREENERS

*****Accessible Processors Variables Assignment List:
Variable Name: => Assigned Value:

V_Local_Screener => P3
V_Sorter_1 => P10
V_Sorter_2 => P1
V_Navigation_1 => P11
V_Navigation_2 => P12
V_Navigation_3 => P13
V_First_Tracker_1 => P4
V_First_Tracker_2 => P5
V_First_Tracker_3 => P8
V_Second_Tracker_1 => P6
V_Second_Tracker_2 => P7
V_Second_Tracker_3 => P9

*****Accessible Memories Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Screen_Memory	=> M2
V_First_Track_Memory	=> M4
V_Second_Track_Memory	=> M5

*****Accessible Busses Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Sort_I/O	=> B3
V_NIOC_I/O	=> B7
V_First_Track_I/O	=> B1
V_Second_Track_I/O	=> B2

**Processor Class Name: SORTERS

***Accessible Processors Variables List:

Variable Name:	Class Restriction:
V_Local_Sorter	SORTERS
V_Screener_1	SCREENERS
V_Screener_2	SCREENERS
V_Navigation_1	NAVIGATIONAL CONTROLLERS
V_Navigation_2	NAVIGATIONAL CONTROLLERS
V_Navigation_3	NAVIGATIONAL CONTROLLERS

***Accessible Memories Variables List:

Variable Name:	Class Restriction:
V_Sort_Memory	SORT MEMORY

***Accessible Busses Variables List:

Variable Name:	Class Restriction
V_Screen_I/O	SORT TO SCREEN
V_NIOC_I/O	SORT WITH NIOC

***Processor Class Instantiation List:

****Processor Name: P1
Processor Class: SORTERS

*****Accessible Processors Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Local_Sorter	=> P10
V_Screener_1	=> P2
V_Screener_2	=> P3
V_Navigation_1	=> P11
V_Navigation_2	=> P12
V_Navigation_3	=> P13

*****Accessible Memories Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Sort_Memory	=> M1

*****Accessible Busses Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Screen_I/O	=> B3
V_NIOC_I/O	=> B4

****Processor Name: P10
Processor Class: SORTERS

*****Accessible Processors Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Local_Sorter	=> P1
V_Screener_1	=> P2
V_Screener_2	=> P3
V_Navigation_1	=> P11
V_Navigation_2	=> P12
V_Navigation_3	=> P13

*****Accessible Memories Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Sort_Memory	=> M1

*****Accessible Busses Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Screen_I/O	=> B3
V_NIOC_I/O	=> B4

**Processor Class Name: TRACKERS

***Accessible Processors Variables List:

Variable Name:	Class Restriction:
V_Local_Tracker_1	TRACKERS
V_Local_Tracker_2	TRACKERS
V_Screener_1	SCREENERS
V_Screener_2	SCREENERS
V_Navigation_1	NAVIGATIONAL CONTROLLERS
V_Navigation_2	NAVIGATIONAL CONTROLLERS
V_Navigation_3	NAVIGATIONAL CONTROLLERS

***Accessible Memories Variables List:

Variable Name:	Class Restriction:
V_Track_Memory	TRACK MEMORY

***Accessible Busses Variables List:

Variable Name:	Class Restriction
V_Screen_I/O	SCREEN WITH TRACK
V_NIOC_I/O	TRACK WITH NIOC

***Processor Class Instantiation List:

****Processor Name: P9
Processor Class: TRACKERS

*****Accessible Processors Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Local_Tracker_1	=> P6
V_Local_Tracker_2	=> P7
V_Screener_1	=> P2
V_Screener_2	=> P3
V_Navigation_1	=> P11
V_Navigation_2	=> P12
V_Navigation_3	=> P13

*****Accessible Memories Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Track_Memory	=> M5

*****Accessible Busses Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Screen_I/O	=> B2
V_NIOC_I/O	=> B6

****Processor Name: P7
Processor Class: TRACKERS

*****Accessible Processors Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Local_Tracker_1	=> P6
V_Local_Tracker_2	=> P9
V_Screener_1	=> P2
V_Screener_2	=> P3
V_Navigation_1	=> P11
V_Navigation_2	=> P12
V_Navigation_3	=> P13

*****Accessible Memories Variables Assignment List:
Variable Name: => Assigned Value:

V_Track_Memory => M5

*****Accessible Busses Variables Assignment List:
Variable Name: => Assigned Value:

V_Screen_I/O => B2
V_NIOC_I/O => B6

****Processor Name: P6
Processor Class: TRACKERS

*****Accessible Processors Variables Assignment List:
Variable Name: => Assigned Value:

V_Local_Tracker_1 => P7
V_Local_Tracker_2 => P9
V_Screener_1 => P2
V_Screener_2 => P3
V_Navigation_1 => P11
V_Navigation_2 => P12
V_Navigation_3 => P13

*****Accessible Memories Variables Assignment List:
Variable Name: => Assigned Value:

V_Track_Memory => M5

*****Accessible Busses Variables Assignment List:
Variable Name: => Assigned Value:

V_Screen_I/O => B2
V_NIOC_I/O => B6

****Processor Name: P8
Processor Class: TRACKERS

*****Accessible Processors Variables Assignment List:
Variable Name: => Assigned Value:

V_Local_Tracker_1 => P4
V_Local_Tracker_2 => P5
V_Screener_1 => P2
V_Screener_2 => P3
V_Navigation_1 => P11
V_Navigation_2 => P12
V_Navigation_3 => P13

*****Accessible Memories Variables Assignment List:
Variable Name: => Assigned Value:

V_Track_Memory => M4

*****Accessible Busses Variables Assignment List:
Variable Name: => Assigned Value:

V_Screen_I/O => B1
V_NIOC_I/O => B5

****Processor Name: P5
Processor Class: TRACKERS

*****Accessible Processors Variables Assignment List:
Variable Name: => Assigned Value:

V_Local_Tracker_1 => P4
V_Local_Tracker_2 => P8
V_Screener_1 => P2
V_Screener_2 => P3
V_Navigation_1 => P11
V_Navigation_2 => P12
V_Navigation_3 => P13

*****Accessible Memories Variables Assignment List:
Variable Name: => Assigned Value:

V_Track_Memory => M4

*****Accessible Busses Variables Assignment List:
Variable Name: => Assigned Value:

V_Screen_I/O => B1
V_NIOC_I/O => B5

****Processor Name: P4
Processor Class: TRACKERS

*****Accessible Processors Variables Assignment List:
Variable Name: => Assigned Value:

V_Local_Tracker_1 => P5
V_Local_Tracker_2 => P8
V_Screener_1 => P2
V_Screener_2 => P3
V_Navigation_1 => P11
V_Navigation_2 => P12
V_Navigation_3 => P13

*****Accessible Memories Variables Assignment List:
Variable Name: => Assigned Value:

V_Track_Memory => M4

*****Accessible Busses Variables Assignment List:
Variable Name: => Assigned Value:

V_Screen_I/O => B1
V_NIOC_I/O => B5

*Memory Class List:

**Memory Class Name: NIOC MEMORY
Size: 8000000 Bytes

I/O Band Width: 16000000 Bytes/Second

***Client Processors Variables List:

Variable Name:	Class Restriction:
V_Local_NIOC_1	NAVIGATIONAL CONTROLLERS
V_Local_NIOC_2	NAVIGATIONAL CONTROLLERS
V_Lccal_NIOC_3	NAVIGATIONAL CONTROLLERS

***Memory Class Instantiation List:

****Memory Name: M3
Memory Class: NIOC MEMORY

*****Client Processors Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Local_NIOC_1	=> P11
V_Local_NIOC_2	=> P12
V_Local_NIOC_3	=> P13

**Memory Class Name: SCREEN MEMORY
Size: 8000000 Byies
I/O Band Width: 16000000 Bytes/Second

***Client Processors Variables List:

Variable Name:	Class Restriction:
V_Local_Screener_1	SCREENERS
V_Local_Screener_2	SCREENERS
V_Navigation_1	NAVIGATIONAL CONTROLLERS
V_Navigation_2	NAVIGATIONAL CONTROLLERS
V_Navigation_3	NAVIGATIONAL CONTROLLERS

***Memory Class Instantiation List:

****Memory Name: M2
Memory Class: SCREEN MEMORY

*****Client Processors Variables Assignment List:

Variable Name: => Assigned Value:

V_Local_Screener_1 => P2
V_Local_Screener_2 => P3
V_Navigation_1 => P11
V_Navigation_2 => P12
V_Navigation_3 => P13

**Memory Class Name: SORT MEMORY
Size: 8000000 Bytes
I/O Band Width: 16000000 Bytes/Second

***Client Processors Variables List:
Variable Name: Class Restriction:

V_Local_Sorter_1 SORTERS
V_Local_Sorter_2 SORTERS
V_Navigation_1 NAVIGATIONAL CONTROLLERS
V_Navigation_2 NAVIGATIONAL CONTROLLERS
V_Navigation_3 NAVIGATIONAL CONTROLLERS

***Memory Class Instantiation List:

****Memory Name: M1
Memory Class: SORT MEMORY

*****Client Processors Variables Assignment List:
Variable Name: => Assigned Value:

V_Local_Sorter_1 => P10
V_Local_Sorter_2 => P1
V_Navigation_1 => P11
V_Navigation_2 => P12
V_Navigation_3 => P13

**Memory Class Name: TRACK MEMORY
Size: 8000000 Bytes
I/O Band Width: 16000000 Bytes/Second

***Client Processors Variables List:

Variable Name:	Class Restriction:
V_Local_Tracker_1	TRACKERS
V_Local_Tracker_2	TRACKERS
V_Local_Tracker_3	TRACKERS
V_Screener_1	SCREENERS
V_Screener_2	SCREENERS

***Memory Class Instantiation List:

****Memory Name: M5
Memory Class: TRACK MEMORY

*****Client Processors Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Local_Tracker_1	=> P6
V_Local_Tracker_2	=> P7
V_Local_Tracker_3	=> P9
V_Screener_1	=> P2
V_Screener_2	=> P3

****Memory Name: M4
Memory Class: TRACK MEMORY

*****Client Processors Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Local_Tracker_1	=> P4
V_Local_Tracker_2	=> P5
V_Local_Tracker_3	=> P8
V_Screener_1	=> P2
V_Screener_2	=> P3

*Bus Class List:

**Bus Class Name: SCREEN WITH NIOC
Effective Band Width: 12000000 Bytes/Second

***Client Processors Variables List:

Variable Name: Class Restriction:

Variable Name	Class Restriction
V_Screener_1	SCREENERS
V_Screener_2	SCREENERS
V_NIOC_1	NAVIGATIONAL CONTROLLERS
V_NIOC_2	NAVIGATIONAL CONTROLLERS
V_NIOC_3	NAVIGATIONAL CONTROLLERS

***Bus Class Instantiation List:

****Bus Name: B7
Bus Class: SCREEN WITH NIOC

*****Client Processors Variables Assignment List:
Variable Name: => Assigned Value:

V_Screener_1 => P2
V_Screener_2 => P3
V_NIOC_1 => P11
V_NIOC_2 => P12
V_NIOC_3 => P13

**Bus Class Name: SORT WITH NIOC
Effective Band Width: 12000000 Bytes/Second

***Client Processors Variables List:
Variable Name: Class Restriction:

V_Sorter_1 SORTERS
V_Sorter_2 SORTERS
V_NIOC_1 NAVIGATIONAL CONTROLLERS
V_NIOC_2 NAVIGATIONAL CONTROLLERS
V_NIOC_3 NAVIGATIONAL CONTROLLERS

***Bus Class Instantiation List:

****Bus Name: B4
Bus Class: SORT WITH NIOC

*****Client Processors Variables Assignment List:

Variable Name: => Assigned Value:

V_Sorter_1 => P10
V_Sorter_2 => P1
V_NIOC_1 => P11
V_NIOC_2 => P12
V_NIOC_3 => P13

**Bus Class Name: SORT TO SCREEN
Effective Band Width: 12000000 Bytes/Second

***Client Processors Variables List:
Variable Name: Class Restriction:

V_Sorter_1 SORTERS
V_Sorter_2 SORTERS
V_Screener_1 SCREENERS
V_Screener_2 SCREENERS

***Bus Class Instantiation List:

****Bus Name: B3
Bus Class: SORT TO SCREEN

*****Client Processors Variables Assignment List:
Variable Name: => Assigned Value:

V_Sorter_1 => P10
V_Sorter_2 => P1
V_Screener_1 => P2
V_Screener_2 => P3

**Bus Class Name: TRACK WITH NIOC
Effective Band Width: 12000000 Bytes/Second

***Client Processors Variables List:
Variable Name: Class Restriction:

V_Tracker_1 TRACKERS
V_Tracker_2 TRACKERS

V_Tracker_3	TRACKERS
V_NIOC_1	NAVIGATIONAL CONTROLLERS
V_NIOC_2	NAVIGATIONAL CONTROLLERS
V_NIOC_3	NAVIGATIONAL CONTROLLERS

***Bus Class Instantiation List:

****Bus Name: B6
Bus Class: TRACK WITH NIOC

*****Client Processors Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Tracker_1	=> P6
V_Tracker_2	=> P7
V_Tracker_3	=> P9
V_NIOC_1	=> P11
V_NIOC_2	=> P12
V_NIOC_3	=> P13

****Bus Name: B5
Bus Class: TRACK WITH NIOC

*****Client Processors Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Tracker_1	=> P4
V_Tracker_2	=> P5
V_Tracker_3	=> P8
V_NIOC_1	=> P11
V_NIOC_2	=> P12
V_NIOC_3	=> P13

**Bus Class Name: SCREEN WITH TRACK
Effective Band Width: 12000000 Bytes/Second

***Client Processors Variables List:

Variable Name:	Class Restriction:
V_Screener_1	SCREENERS

V_Screener_2	SCREENERS
V_Tracker_1	TRACKERS
V_Tracker_2	TRACKERS
V_Tracker_3	TRACKERS

***Bus Class Instantiation List:

****Bus Name: B2
Bus Class: SCREEN WITH TRACK

*****Client Processors Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Screener_1	=> P2
V_Screener_2	=> P3
V_Tracker_1	=> P6
V_Tracker_2	=> P7
V_Tracker_3	=> P9

****Bus Name: B1
Bus Class: SCREEN WITH TRACK

*****Client Processors Variables Assignment List:

Variable Name:	=> Assigned Value:
V_Screener_1	=> P2
V_Screener_2	=> P3
V_Tracker_1	=> P4
V_Tracker_2	=> P5
V_Tracker_3	=> P8

*Task List:

Task Name: AOA/AOSP Tracking
Task Class Name: AOSP Tracking
Processor Ensemble Name: AOSP Tracking
Task Minimum Start Time: 0.000000E+00
Task Data Set Size: 50
Task Class Author: TBE
Task Class Creation Date: 07/19/89

Task Class Validation Date: 08/10/89
Task Class Validation Time: 13:38:27.07

**Input Start Time Dependency Variables Assignments List:
Variable Name: ==> Assigned Value:

--- List Empty ---

**Output Start Time Dependency Variables Assignments List:
Variable Name: ==> Assigned Value:

--- List Empty ---

**Segment Class Table

***Segment Class Name: ANGULAR RATE SMOOTHING LAG
Target Processor Class: SCREENERS
Number of Instantiations: 1
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 5.800000E-06 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 3.840000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 3.840000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 3.840000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 3.840000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 3.840000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 3.840000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Sort_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O

Bus Variable Class Restriction: SCREEN WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_First_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 6.900000E-01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Second_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 6.900000E-01 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: INITIALIZATION 3
Target Processor Class: NAVIGATIONAL CONTROLLERS
Number of Instantiations: 1
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_NIOC_Memory
Memory Variable Class Restriction: NIOC MEMORY
R = 1.000000E+00 Q1 = 1.994540E+05 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_NIOC_Memory
Memory Variable Class Restriction: NIOC MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Sort_I/O
Bus Variable Class Restriction: SORT WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_First_Track_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Second_Track_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: NAVIGATION UPDATE
Target Processor Class: NAVIGATIONAL CONTROLLERS
Number of Instantiations: 1
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 4.999450E-01 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_NIOC_Memory
Memory Variable Class Restriction: NIOC MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_NIOC_Memory
Memory Variable Class Restriction: NIOC MEMORY
R = 1.000000E+00 Q1 = 1.950800E+04 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Sort_I/O
Bus Variable Class Restriction: SORT WITH NIOC
R = 1.000000E+00 Q1 = 2.100000E+03 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_First_Track_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 4.200000E+03 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Second_Track_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 4.200000E+03 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: INITIALIZATION 1
Target Processor Class: SORTERS
Number of Instantiations: 1
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 1.528620E+05 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: SORT WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: MEASUREMENT PROCESSING LAG JOIN
Target Processor Class: SORTERS
Number of Instantiations: 1
Segment Class Type: Join

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: SORT WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: MEASUREMENT PROCESSING LAG
Target Processor Class: SORTERS
Number of Instantiations: 1
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:

R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 1.191800E-03 L1 = 0.000000E+00
Q2 = 2.080000E-05 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = -6.900000E+01 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 6.900000E+01 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00 Q1 = 2.600000E+01 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: SORT WITH NIOC
R = 1.000000E+00 Q1 = 5.400000E+01 L1 = 0.000000E+00
Q2 = 3.600000E-02 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: MEASUREMENT PROCESSING
Target Processor Class: SORTERS
Number of Instantiations: 1
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 2.264330E-02 L1 = 0.000000E+00
Q2 = 3.952000E-04 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = -1.311000E+03 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 1.974300E+04 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: SORT WITH NIOC
R = 1.000000E+00 Q1 = 1.026000E+03 L1 = 0.000000E+00
Q2 = 6.840000E-01 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: REFERENCE STAR MATCHING JOIN
Target Processor Class: NAVIGATIONAL CONTROLLERS
Number of Instantiations: 1
Segment Class Type: Join

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_NIOC_Memory
Memory Variable Class Restriction: NIOC MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_NIOC_Memory
Memory Variable Class Restriction: NIOC MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Sort_I/O
Bus Variable Class Restriction: SORT WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_First_Track_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Second_Track_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: REFERENCE STAR MATCHING
Target Processor Class: NAVIGATIONAL CONTROLLERS
Number of Instantiations: 1
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 6.000000E+00 L1 = 0.000000E+00
Q2 = 4.000000E-03 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_NIOC_Memory
Memory Variable Class Restriction: NIOC MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Screen_Memory

Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_NIOC_Memory
Memory Variable Class Restriction: NIOC MEMORY
R = 1.000000E+00 Q1 = 2.969600E+04 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Sort_I/O
Bus Variable Class Restriction: SORT WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_First_Track_I/O
Bus Variable Class Restriction: TRACK WITH NIOC

R = 1.000000E+00 Q1 = 4.000000E+01 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Second_Track_I/O
Bus Variabie Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 4.000000E+01 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: OBJECT SORTING LAG JOIN
Target Processor Class: SORTERS
Number of Instantiations: 1
Segment Class Type: Join

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: SORT WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: OBJECT SORTING LAG
Target Processor Class: SORTERS
Number of Instantiations: 1
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 1.000000E-06 L1 = 0.000000E+00
Q2 = 5.200000E-06 L2 = 0.000000E+00
Q3 = 9.000000E-07

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = -2.300000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 2.300000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00 Q1 = 2.000000E+00 L1 = 0.000000E+00
Q2 = 2.300000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: SORT WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: OBJECT SORTING
Target Processor Class: SORTERS
Number of Instantiations: 1
Segment Class Type: Application Code

***Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 1.900000E-05 L1 = 0.000000E+00
Q2 = 9.900000E-05 L2 = 0.000000E+00
Q3 = 1.800000E-07

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00   Q1 = 0.000000E+00   L1 = 0.000000E+00
Q2 = -4.370000E+01   L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00   Q1 = 1.228800E+04   L1 = 0.000000E+00
Q2 = 4.370000E+01   L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00   Q1 = 1.760000E+02   L1 = 0.000000E+00
Q2 = 4.370000E+01   L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: SORT WITH NIOC
R = 1.000000E+00   Q1 = 0.000000E+00   L1 = 0.000000E+00
Q2 = 0.000000E+00   L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: INITIALIZATION 2
Target Processor Class: SCREENERS
Number of Instantiations: 1
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:

M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 2.681800E+05 L1 = 0.000000E+00
Q2 = 4.066940E-01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Sort_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: SCREEN WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_First_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Second_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: OBJECT SCREENING LAG JOIN
Target Processor Class: SCREENERS
Number of Instantiations: 1
Segment Class Type: Join

****Transfer Functions List:
N = Data Set Size:

R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Sort_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: SCREEN WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_First_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Second_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: OBJECT SCREENING LAG
Target Processor Class: SCREENERS
Number of Instantiations: 1
Segment Class Type: Application Code

****Transfer Functions List:

N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 1.700000E-07 L1 = 0.000000E+00
Q2 = 9.000000E-07 L2 = 0.000000E+00
Q3 = 4.000000E-07

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = -3.660246E-01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 3.660246E-01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY

Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Sort_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: SCREEN WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_First_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 1.274900E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Second_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 1.274900E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: HANDOVER JOIN
Target Processor Class: NAVIGATIONAL CONTROLLERS
Number of Instantiations: 1
Segment Class Type: Join

****Transfer Functions List:

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_NIOC_Memory
Memory Variable Class Restriction: NIOC MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_NIOC_Memory
Memory Variable Class Restriction: NIOC MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Sort_I/O
Bus Variable Class Restriction: SORT WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_First_Track_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Second_Track_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: HANDOVER
Target Processor Class: NAVIGATIONAL CONTROLLERS
Number of Instantiations: 1

Segment Class Type: Application Code

****Transfer Functions List:

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 9.375000E-04 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_NIOC_Memory
Memory Variable Class Restriction: NIOC MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_NIOC_Memory
Memory Variable Class Restriction: NIOC MEMORY
R = 1.000000E+00 Q1 = 3.020800E+04 L1 = 0.000000E+00
Q2 = 8.958135E+02 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Sort_Memory
Memory Variable Class Restriction: SORT MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Sort_I/O
Bus Variable Class Restriction: SORT WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_First_Track_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Second_Track_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: TRACK FITTING JOIN
Target Processor Class: TRACKERS

Number of Instantiations: 2
Segment Class Type: Join

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O

Bus Variable Class Restriction: TRACK WITH NIOC

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: TRACK FITTING
Target Processor Class: TRACKERS
Number of Instantiations: 2
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 6.354600E-03 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 4.454400E+04 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 3.268080E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: TRACK INITIALIZATION LAG
Target Processor Class: TRACKERS
Number of Instantiations: 1
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 2.880000E-01 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 5.400000E+01 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 1.140000E+02 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: TRACK INITIALIZATION
Target Processor Class: TRACKERS
Number of Instantiations: 1
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = -2.880000E-01 L1 = 0.000000E+00
Q2 = 1.394380E-02 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory

Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 5.785600E+04 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = -5.400000E+01 L1 = 0.000000E+00
Q2 = 2.614464E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = -1.140000E+02 L1 = 0.000000E+00
Q2 = 5.519424E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: RADIOMETRIC INITIALIZATION
Target Processor Class: TRACKERS
Number of Instantiations: 2
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 2.032900E-03 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 4.798989E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 4.352000E+04 L1 = 0.000000E+00
Q2 = 4.798989E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: DESIGNATION INITIALIZATION
Target Processor Class: TRACKERS
Number of Instantiations: 2
Segment Class Type: Application Code

****Transfer Functions List:

N = Data Set Size:

R = Data Set Size Reduction Factor:

M = R * N:

F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 5.072000E-04 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 7.680000E+03 L1 = 0.000000E+00
Q2 = 9.597648E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

***Segment Class Name: PREDICTION INITIALIZATION
Target Processor Class: TRACKERS
Number of Instantiations: 2
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 2.277500E-03 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 9.062400E+04 L1 = 0.000000E+00
Q2 = 4.799013E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 2.114811E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 4.286559E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: CGP LAG JOIN
Target Processor Class: TRACKERS
Number of Instantiations: 2
Segment Class Type: Join

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: CANDIDATE GENERATION PROCESS LAG
Target Processor Class: TRACKERS
Number of Instantiations: 2
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 2.481225E-01 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 7.680000E+02 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 2.232000E+02 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: CANDIDATE GENERATION PROCESS
Target Processor Class: TRACKERS
Number of Instantiations: 2
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = -2.481225E-01 L1 = 0.000000E+00
Q2 = 2.725300E-02 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 4.172800E+04 L1 = 0.000000E+00
Q2 = 1.227571E+02 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = -2.232000E+02 L1 = 0.000000E+00
Q2 = 1.778112E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: CGP TRACK ACCEPT MESSAGE HANDLER
Target Processor Class: TRACKERS
Number of Instantiations: 2
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:

M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 1.920000E-05 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = -5.205683E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 5.205683E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: ANGULAR RATE SMOOTHING LAG JOIN
Target Processor Class: SCREENERS
Number of Instantiations: 1
Segment Class Type: Join

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Sort_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: SCREEN WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_First_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Second_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: TDM INITIAL INITIATOR LOADING
Target Processor Class: SCREENERS
Number of Instantiations: 1
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 6.000000E-04 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: *V_Screen_Memory*
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory

Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Sort_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: SCREEN WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_First_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 6.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Second_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 6.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: ANGULAR RATE SMOOTHING
Target Processor Class: SCREENERS
Number of Instantiations: 1
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 1.093000E-04 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 7.296000E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 7.296000E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 7.296000E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 2.611200E+04 L1 = 0.000000E+00
Q2 = 7.296000E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 7.296000E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 7.296000E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Sort_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: SCREEN WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_First_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 1.311000E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Second_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 1.311000E+01 L2 = 0.000000E+00

Q3 = 0.000000E+00

***Segment Class Name: TDM BUILD CAND TRACK MSG JOIN 1
Target Processor Class: SCREENERS
Number of Instantiations: 1
Segment Class Type: Join

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Sort_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: SCREEN WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_First_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Second_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: TDM BUILD CAND TRACK MSG JOIN 2
Target Processor Class: SCREENERS
Number of Instantiations: 1
Segment Class Type: Join

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Sort_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: SCREEN WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_First_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Second_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: TDM BUILD CANDIDATE TRACK MSG
Target Processor Class: SCREENERS
Number of Instantiations: 1
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 3.430000E-05 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 9.295872E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Sort_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: SCREEN WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_First_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 3.340704E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Second_Track_I/O

Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 3.340704E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: TDM REMAINING INITIATOR LOADING
Target Processor Class: SCREENERS
Number of Instantiations: 1
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = -6.000000E-04 L1 = 0.000000E+00
Q2 = 2.160000E-05 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Sort_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: SCREEN WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_First_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = -6.000000E+00 L1 = 0.000000E+00
Q2 = 2.155200E-01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Second_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = -6.000000E+00 L1 = 0.000000E+00
Q2 = 2.155200E-01 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: TDM TRACK ACCEPT/REJECT MSG HAND
Target Processor Class: SCREENERS
Number of Instantiations: 1
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 8.810000E-05 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = -5.045910E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 3.174400E+04 L1 = 0.000000E+00
Q2 = 5.365456E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Sort_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: SCREEN WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_First_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 5.228928E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_Second_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 5.228928E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: PREDICTED WINDOW FILE SORT
Target Processor Class: SCREENERS
Number of Instantiations: 1
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 9.000000E-06 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 7.320492E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00

Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Screen_Memory
Memory Variable Class Restriction: SCREEN MEMORY
R = 1.000000E+00 Q1 = 7.168000E+03 L1 = 0.000000E+00
Q2 = 7.320492E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_First_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Variable Name: V_Second_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Sort_I/O
Bus Variable Class Restriction: SORT TO SCREEN
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: SCREEN WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_First_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00

Q3 = 0.000000E+00

Bus Variable Name: V_Second_Track_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: TRACK INITIALIZATION (ALL)
Target Processor Class: TRACKERS
Number of Instantiations: 1
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 1.394380E-02 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 5.785600E+04 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 2.614464E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 5.519424E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: INITIALIZATION 4 & 5
Target Processor Class: TRACKERS
Number of Instantiations: 2
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 9.660000E+05 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00   Q1 = 0.000000E+00   L1 = 0.000000E+00
Q2 = 0.000000E+00   L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00   Q1 = 0.000000E+00   L1 = 0.000000E+00
Q2 = 0.000000E+00   L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00   Q1 = 0.000000E+00   L1 = 0.000000E+00
Q2 = 0.000000E+00   L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: TRACK UPDATE JOIN
Target Processor Class: TRACKERS
Number of Instantiations: 2
Segment Class Type: Join

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00   Q1 = 0.000000E+00   L1 = 0.000000E+00
Q2 = 0.000000E+00   L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: TRACK UPDATE
Target Processor Class: TRACKERS
Number of Instantiations: 2
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 2.026100E-03 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = -2.100000E-02 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 4.352000E+04 L1 = 0.000000E+00
Q2 = 3.353560E+02 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 1.330000E+01 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: RADIOMETRIC UPDATE
Target Processor Class: TRACKERS

Number of Instantiations: 2
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 2.446200E-03 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 3.532800E+04 L1 = 0.000000E+00
Q2 = 3.304000E+02 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O

Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: DESIGNATION UPDATE
Target Processor Class: TRACKERS
Number of Instantiations: 2
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 1.221200E-03 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 7.680000E+03 L1 = 0.000000E+00
Q2 = 3.304000E+02 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

***Segment Class Name: PREDICTION UPDATE
Target Processor Class: TRACKERS
Number of Instantiations: 2
Segment Class Type: Application Code

****Transfer Functions List:
N = Data Set Size:
R = Data Set Size Reduction Factor:
M = R * N:
F(N) = G(M) = Q1 + (Q2 * M) + (Q3 * M**2) + (L1 +(L2 * M))LOG2(M)

Run Time Transfer Function Coeficients:

R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 1.459330E-02 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory Space Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00 Q1 = 0.000000E+00 L1 = 0.000000E+00
Q2 = 0.000000E+00 L2 = 0.000000E+00
Q3 = 0.000000E+00

Memory I/O Requirements Transfer Function Coeficients:

Memory Variable Name: V_Track_Memory
Memory Variable Class Restriction: TRACK MEMORY
R = 1.000000E+00   Q1 = 9.062400E+04   L1 = 0.000000E+00
Q2 = 1.486800E+02   L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus I/O Requirements Transfer Function Coeficients:

Bus Variable Name: V_Screen_I/O
Bus Variable Class Restriction: SCREEN WITH TRACK
R = 1.000000E+00   Q1 = 0.000000E+00   L1 = 0.000000E+00
Q2 = 7.288000E+01   L2 = 0.000000E+00
Q3 = 0.000000E+00

Bus Variable Name: V_NIOC_I/O
Bus Variable Class Restriction: TRACK WITH NIOC
R = 1.000000E+00   Q1 = 0.000000E+00   L1 = 0.000000E+00
Q2 = 1.475600E+01   L2 = 0.000000E+00
Q3 = 0.000000E+00

**Thread Table

***Thread Name: THD_12: HO
Target Processor: P12
Target Processor Class: NAVIGATIONAL CONTROLLERS
****Segment List:

Segment Name: Segment 40 Join
Segment Class: HANDOVER JOIN
Segment Type: Join
Predecessor Segment: Segment 42
Predecessor Thread: THD_7: TI2

Segment Name: Segment 40
Segment Class: HANDOVER
Segment Type: Application Code

***Thread Name: THD_7: TI2
Target Processor: P7
Target Processor Class: TRACKERS

****Segment List:

Segment Name: Segment 26 Join
Segment Class: TRACK FITTING JOIN
Segment Type: Join
Predecessor Segment: Segment 23
Predecessor Thread: THD_6: CGP2

Segment Name: Segment 26
Segment Class: TRACK FITTING
Segment Type: Application Code

Segment Name: Segment 42
Segment Class: TRACK INITIALIZATION LAG
Segment Type: Application Code

Segment Name: Segment 27
Segment Class: TRACK INITIALIZATION
Segment Type: Application Code

Segment Name: Segment 28
Segment Class: RADIOMETRIC INITIALIZATION
Segment Type: Application Code

Segment Name: Segment 29
Segment Class: DESIGNATION INITIALIZATION
Segment Type: Application Code

Segment Name: Segment 30
Segment Class: PREDICTION INITIALIZATION
Segment Type: Application Code

***Thread Name: THD_5: TI1
Target Processor: P5
Target Processor Class: TRACKERS
****Segment List:

Segment Name: Segment 14 Join
Segment Class: TRACK FITTING JOIN
Segment Type: Join
Predecessor Segment: Segment 11
Predecessor Thread: THD_4: CGP1

Segment Name: Segment 14
Segment Class: TRACK FITTING
Segment Type: Application Code

Segment Name: Segment 15
Segment Class: TRACK INITIALIZATION (ALL)
Segment Type: Application Code

Segment Name: Segment 16
Segment Class: RADIOMETRIC INITIALIZATION
Segment Type: Application Code

Segment Name: Segment 17
Segment Class: DESIGNATION INITIALIZATION
Segment Type: Application Code

Segment Name: Segment 18
Segment Class: PREDICTION INITIALIZATION
Segment Type: Application Code

***Thread Name: THD_3: ARS/TDM
Target Processor: P3
Target Processor Class: SCREENERS
****Segment List:

Segment Name: Segment 35 Join
Segment Class: ANGULAR RATE SMOOTHING LAG JOIN
Segment Type: Join
Predecessor Segment: Segment 3
Predecessor Thread: THD_2: OSC

Segment Name: Segment 35
Segment Class: ANGULAR RATE SMOOTHING LAG
Segment Type: Application Code

Segment Name: Segment 6
Segment Class: TDM INITIAL INITIATOR LOADING
Segment Type: Application Code

Segment Name: Segment 5
Segment Class: ANGULAR RATE SMOOTHING

Segment Type: Application Code

Segment Name: Segment 7 Join 1
Segment Class: TDM BUILD CAND TRACK MSG JOIN 1
Segment Type: Join
Predecessor Segment: Segment 23
Predecessor Thread: THD_6: CGP2

Segment Name: Segment 7 Join 2
Segment Class: TDM BUILD CAND TRACK MSG JOIN 2
Segment Type: Join
Predecessor Segment: Segment 11
Predecessor Thread: THD_4: CGP1

Segment Name: Segment 7
Segment Class: TDM BUILD CANDIDATE TRACK MSG
Segment Type: Application Code

Segment Name: Segment 8
Segment Class: TDM REMAINING INITIATOR LOADING
Segment Type: Application Code

Segment Name: Segment 9
Segment Class: TDM TRACK ACCEPT/REJECT MSG HAND
Segment Type: Application Code

Segment Name: Segment 10
Segment Class: PREDICTED WINDOW FILE SORT
Segment Type: Application Code

***Thread Name: THD_6: CGP2
Target Processor: P6
Target Processor Class: TRACKERS
****Segment List:

Segment Name: Segment 23 Join
Segment Class: CGP LAG JOIN
Segment Type: Join
Predecessor Segment: Segment 6
Predecessor Thread: THD_3: ARS/TDM

Segment Name: Segment 23

Segment Class: CANDIDATE GENERATION PROCESS LAG
Segment Type: Application Code

Segment Name: Segment 24
Segment Class: CANDIDATE GENERATION PROCESS
Segment Type: Application Code

Segment Name: Segment 25
Segment Class: CGP TRACK ACCEPT MESSAGE HANDLER
Segment Type: Application Code

***Thread Name: THD_4: CGP1
Target Processor: P4
Target Processor Class: TRACKERS
****Segment List:

Segment Name: Segment 11 Join
Segment Class: CGP LAG JOIN
Segment Type: Join
Predecessor Segment: Segment 6
Predecessor Thread: THD_3: ARS/TDM

Segment Name: Segment 11
Segment Class: CANDIDATE GENERATION PROCESS LAG
Segment Type: Application Code

Segment Name: Segment 12
Segment Class: CANDIDATE GENERATION PROCESS
Segment Type: Application Code

Segment Name: Segment 13
Segment Class: CGP TRACK ACCEPT MESSAGE HANDLER
Segment Type: Application Code

***Thread Name: THD_8: TU1
Target Processor: P8
Target Processor Class: TRACKERS
****Segment List:

Segment Name: Segment 45
Segment Class: INITIALIZATION 4 & 5

Segment Type: Application Code

Segment Name: Segment 19 Join
Segment Class: TRACK UPDATE JOIN
Segment Type: Join
Predecessor Segment: Segment 3
Predecessor Thread: THD_2: OSC

Segment Name: Segment 19
Segment Class: TRACK UPDATE
Segment Type: Application Code

Segment Name: Segment 20
Segment Class: RADIOMETRIC UPDATE
Segment Type: Application Code

Segment Name: Segment 21
Segment Class: DESIGNATION UPDATE
Segment Type: Application Code

Segment Name: Segment 22
Segment Class: PREDICTION UPDATE
Segment Type: Application Code

***Thread Name: THD_9: TU2
Target Processor: P9
Target Processor Class: TRACKERS
****Segment List:

Segment Name: Segment 46
Segment Class: INITIALIZATION 4 & 5
Segment Type: Application Code

Segment Name: Segment 31 Join
Segment Class: TRACK UPDATE JOIN
Segment Type: Join
Predecessor Segment: Segment 3
Predecessor Thread: THD_2: OSC

Segment Name: Segment 31
Segment Class: TRACK UPDATE
Segment Type: Application Code

Segment Name: Segment 32
Segment Class: RADIOMETRIC UPDATE
Segment Type: Application Code

Segment Name: Segment 33
Segment Class: DESIGNATION UPDATE
Segment Type: Application Code

Segment Name: Segment 34
Segment Class: PREDICTION UPDATE
Segment Type: Application Code

***Thread Name: THD_2: OSC
Target Processor: P2
Target Processor Class: SCREENERS
****Segment List:

Segment Name: Segment 44
Segment Class: INITIALIZATION 2
Segment Type: Application Code

Segment Name: Segment 3 Join
Segment Class: OBJECT SCREENING LAG JOIN
Segment Type: Join
Predecessor Segment: Segment 1
Predecessor Thread: THD_1: OSO

Segment Name: Segment 3
Segment Class: OBJECT SCREENING LAG
Segment Type: Application Code

Segment Name: Segment 4
Segment Class: OBJECT SCREENING
Segment Type: Application Code

***Thread Name: THD_1: OSO
Target Processor: P1
Target Processor Class: SORTERS
****Segment List:

Segment Name: Segment 1 Join
Segment Class: OBJECT SORTING LAG JOIN
Segment Type: Join
Predecessor Segment: Segment 36
Predecessor Thread: THD_10: MP

Segment Name: Segment 1
Segment Class: OBJECT SORTING LAG
Segment Type: Application Code

Segment Name: Segment 2
Segment Class: OBJECT SORTING
Segment Type: Application Code

***Thread Name: THD_13: RSM
Target Processor: P13
Target Processor Class: NAVIGATIONAL CONTROLLERS
****Segment List:

Segment Name: Segment 41 Join
Segment Class: REFERENCE STAR MATCHING JOIN
Segment Type: Join
Predecessor Segment: Segment 36
Predecessor Thread: THD_10: MP

Segment Name: Segment 41
Segment Class: REFERENCE STAR MATCHING
Segment Type: Application Code

***Thread Name: THD_10: MP
Target Processor: P10
Target Processor Class: SORTERS
****Segment List:

Segment Name: Segment 43
Segment Class: INITIALIZATION 1
Segment Type: Application Code

Segment Name: Segment 36 Join
Segment Class: MEASUREMENT PROCESSING LAG JOIN
Segment Type: Join

Predecessor Segment: Segment 38
Predecessor Thread: THD_11: NAV

Segment Name: Segment 36
Segment Class: MEASUREMENT PROCESSING LAG
Segment Type: Application Code

Segment Name: Segment 37
Segment Class: MEASUREMENT PROCESSING
Segment Type: Application Code

***Thread Name: THD_11: NAV
Target Processor: P11
Target Processor Class: NAVIGATIONAL CONTROLLERS
****Segment List:

Segment Name: Segment 38
Segment Class: INITIALIZATION 3
Segment Type: Application Code

Segment Name: Segment 39
Segment Class: NAVIGATION UPDATE
Segment Type: Application Code

APPENDIX D

PROCESSOR 3 EVENT ACTIVITIES

****

***** PROCESSOR EVENT ACTIVITIES *****

Processor Profile for: P3
Event Listing from file : Track.ehf
Processor Ensemble : AOSP Tracking
Author: TBE
Load Duration: 6.20223
Event Listing Start Time: 0
Event Listing End Time : 6.20223

****

SEGMENT EVENT for JOIN SEGMENT
Time = 0 Duration = 0
Segment: Segment 35 Join of Class: ANGULAR RATE SMOOTHING LAG JOIN
Task: AOA/AOSP Tracking of Class: AOSP Tracking
Thread: THD_3: ARS/TDM
Processor: P3
of Class: SCREENERS
RESOURCE USE PROFILE AT THE SEGMENT BOUNDARY EVENT
Memory: M2
of Class: SCREEN MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M2
of Class: SCREEN MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M4
of Class: TRACK MEMORY

MEMORY I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B3
of Class: SORT TO SCREEN
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B7
of Class: SCREEN WITH NIOC
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B1
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B2
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
TOTAL TIME PERCENT
SEGMENT EVENT for JOIN SEGMENT
Time = 0 Duration = 0.00578797
Segment: Segment 35 Join of Class: ANGULAR RATE SMOOTHING LAG JOIN
Task: AOA/AOSP Tracking of Class: AOSP Tracking
Thread: THD_3: ARS/TDM
Processor: P3
of Class: SCREENERS
RESOURCE USE PROFILE AT THE SEGMENT BOUNDARY EVENT

Memory: M2
of Class: SCREEN MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M2
of Class: SCREEN MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B3
of Class: SORT TO SCREEN
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B7
of Class: SCREEN WITH NIOC
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B1
of Class: SCREEN WITH TRACK
BUS I/O

Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B2
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
TOTAL TIME PERCENT
SEGMENT EVENT for APPLICATION CODE SEGMENT
Time = 0.00578797 Duration = 0.00029
Segment: Segment 35 of Class: ANGULAR RATE SMOOTHING LAG
Task: AOA/AOSP Tracking of Class: AOSP Tracking
Thread: THD_3: ARS/TDM
Processor: P3
of Class: SCREENERS
RESOURCE USE PROFILE AT THE SEGMENT BOUNDARY EVENT
Memory: M2
of Class: SCREEN MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M2
of Class: SCREEN MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 4.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 4.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M5

of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 4.000000%
Bus: B3
of Class: SORT TO SCREEN
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B7
of Class: SCREEN WITH NIOC
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B1
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B2
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
TOTAL TIME PERCENT
SEGMENT EVENT for APPLICATION CODE SEGMENT
Time = 0.00607797 Duration = 0.0006
Segment: Segment 6 of Class: TDM INITIAL INITIATOR
LOADING
Task: AOA/AOSP Tracking of Class: AOSP Tracking
Thread: THD_3: ARS/TDM
Processor: P3
of Class: SCREENERS
RESOURCE USE PROFILE AT THE SEGMENT BOUNDARY EVENT
Memory: M2
of Class: SCREEN MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M2
of Class: SCREEN MEMORY
MEMORY I/O
Utilization prior 4.000000%

Utilization after 0.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 4.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 4.000000%
Utilization after 0.000000%
Bus: B3
of Class: SORT TO SCREEN
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B7
of Class: SCREEN WITH NIOC
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B1
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B2
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
TOTAL TIME PERCENT
SEGMENT EVENT for APPLICATION CODE SEGMENT

Time = 0.00667797 Duration = 0.005465
Segment: Segment 5 of Class: ANGULAR RATE SMOOTHING
Task: AOA/AOSP Tracking of Class: AOSP Tracking
Thread: THD_3: ARS/TDM
Processor: P3
of Class: SCREENERS
RESOURCE USE PROFILE AT THE SEGMENT BOUNDARY EVENT
Memory: M2
of Class: SCREEN MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M2
of Class: SCREEN MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 34.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 4.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 4.000000%
Bus: B3
of Class: SORT TO SCREEN
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B7

of Class: SCREEN WITH NIOC
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B1
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B2
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
TOTAL TIME PERCENT
SEGMENT EVENT for JOIN SEGMENT
Time = 0.012143 Duration = 0
Segment: Segment 7 Join 1 of Class: TDM BUILD CAND TRACK
MSG JOIN 1
Task: AOA/AOSP Tracking of Class: AOSP Tracking
Thread: THD_3: ARS/TDM
Processor: P3
of Class: SCREENERS
RESOURCE USE PROFILE AT THE SEGMENT BOUNDARY EVENT
Memory: M2
of Class: SCREEN MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M2
of Class: SCREEN MEMORY
MEMORY I/O
Utilization prior 34.000000%
Utilization after 0.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 4.000000%

Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 4.000000%
Utilization after 0.000000%
Bus: B3
of Class: SORT TO SCREEN
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B7
of Class: SCREEN WITH NIOC
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B1
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B2
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
TOTAL TIME PERCENT
SEGMENT EVENT for JOIN SEGMENT
Time = 0.012143 Duration = 0.242657
Segment: Segment 7 Join 1 of Class: TDM BUILD CAND TRACK
MSG JOIN 1
Task: AOA/AOSP Tracking of Class: AOSP Tracking
Thread: THD_3: ARS/TDM
Processor: P3
of Class: SCREENERS
RESOURCE USE PROFILE AT THE SEGMENT BOUNDARY EVENT
Memory: M2
of Class: SCREEN MEMORY

MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M2
of Class: SCREEN MEMORY
MEMORY I/O
Utilization prior 34.000000%
Utilization after 0.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 4.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 4.000000%
Utilization after 0.000000%
Bus: B3
of Class: SORT TO SCREEN
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B7
of Class: SCREEN WITH NIOC
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B1
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%

Bus: B2
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
TOTAL TIME PERCENT
SEGMENT EVENT for JOIN SEGMENT
Time = 0.2548 Duration = 0
Segment: Segment 7 Join 2 of Class: TDM BUILD CAND TRACK
MSG JOIN 2
Task: AOA/AOSP Tracking of Class: AOSP Tracking
Thread: THD_3: ARS/TDM
Processor: P3
of Class: SCREENERS
RESOURCE USE PROFILE AT THE SEGMENT BOUNDARY EVENT
Memory: M2
of Class: SCREEN MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M2
of Class: SCREEN MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY I/O

Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B3
of Class: SORT TO SCREEN
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B7
of Class: SCREEN WITH NIOC
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B1
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B2
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
TOTAL TIME PERCENT
SEGMENT EVENT for JOIN SEGMENT
Time = 0.2548 Duration = 0
Segment: Segment 7 Join 2 of Class: TDM BUILD CAND TRACK
MSG JOIN 2
Task: AOA/AOSP Tracking of Class: AOSP Tracking
Thread: THD_3: ARS/TDM
Processor: P3
of Class: SCREENERS
RESOURCE USE PROFILE AT THE SEGMENT BOUNDARY EVENT
Memory: M2
of Class: SCREEN MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M2
of Class: SCREEN MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M4

of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B3
of Class: SORT TO SCREEN
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B7
of Class: SCREEN WITH NIOC
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B1
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B2
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
TOTAL TIME PERCENT
SEGMENT EVENT for APPLICATION CODE SEGMENT
Time = 0.2548 Duration = 0.001715

Segment: Segment 7 of Class: TDM BUILD CANDIDATE
TRACK MSG
Task: AOA/AOSP Tracking of Class: AOSP Tracking
Thread: THD_3: ARS/TDM
Processor: P3
of Class: SCREENERS
RESOURCE USE PROFILE AT THE SEGMENT BOUNDARY EVENT
Memory: M2
of Class: SCREEN MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M2
of Class: SCREEN MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 17.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B3
of Class: SORT TO SCREEN
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B7

of Class: SCREEN WITH NIOC
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B1
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 6.000000%
Bus: B2
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 6.000000%
TOTAL TIME PERCENT
SEGMENT EVENT for APPLICATION CODE SEGMENT
Time = 0.256515 Duration = 0.00048
Segment: Segment 8 of Class: TDM REMAINING INITIATOR LOADING
Task: AOA/AOSP Tracking of Class: AOSP Tracking
Thread: THD_3: ARS/TDM
Processor: P3
of Class: SCREENERS
RESOURCE USE PROFILE AT THE SEGMENT BOUNDARY EVENT
Memory: M2
of Class: SCREEN MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M2
of Class: SCREEN MEMORY
MEMORY I/O
Utilization prior 17.000000%
Utilization after 0.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 0.000000%

Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B3
of Class: SORT TO SCREEN
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B7
of Class: SCREEN WITH NIOC
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B1
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 6.000000%
Utilization after 0.000000%
Bus: B2
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 6.000000%
Utilization after 0.000000%
TOTAL TIME PERCENT
SEGMENT EVENT for APPLICATION CODE SEGMENT
Time = 0.256995 Duration = 0.004405
Segment: Segment 9 of Class: TDM TRACK ACCEPT/REJECT
MSG HAND
Task: AOA/AOSP Tracking of Class: AOSP Tracking
Thread: THD_3: ARS/TDM
Processor: P3
of Class: SCREENERS
RESOURCE USE PROFILE AT THE SEGMENT BOUNDARY EVENT
Memory: M2
of Class: SCREEN MEMORY

MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M2
of Class: SCREEN MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 48.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B3
of Class: SORT TO SCREEN
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B7
of Class: SCREEN WITH NIOC
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B1
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%

Bus: B2
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
TOTAL TIME PERCENT
SEGMENT EVENT for APPLICATION CODE SEGMENT
Time = 0.2614 Duration = 0.00045
Segment: Segment 10 of Class: PREDICTED WINDOW FILE SORT
Task: AOA/AOSP Tracking of Class: AOSP Tracking
Thread: THD_3: ARS/TDM
Processor: P3
of Class: SCREENERS
RESOURCE USE PROFILE AT THE SEGMENT BOUNDARY EVENT
Memory: M2
of Class: SCREEN MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M2
of Class: SCREEN MEMORY
MEMORY I/O
Utilization prior 48.000000%
Utilization after 104.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M4
of Class: TRACK MEMORY
MEMORY I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY CAPACITY
Utilization prior 0.000000%
Utilization after 0.000000%
Memory: M5
of Class: TRACK MEMORY
MEMORY I/O

Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B3
of Class: SORT TO SCREEN
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B7
of Class: SCREEN WITH NIOC
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B1
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
Bus: B2
of Class: SCREEN WITH TRACK
BUS I/O
Utilization prior 0.000000%
Utilization after 0.000000%
TOTAL TIME PERCENT
SEGMENT EVENT for BUS BANDWIDTH
Time = 0.26185 Duration = 0.00045